Testability of Switching Lattices in the Cellular Fault Model

Marie Skłodowska-Curie grant agreement No 691178
(European Union’s Horizon 2020 research and innovation programme)

Anna Bernasconi\(^1\), Valentina Ciriani\(^2\), Luca Frontini\(^3\)

\(^1\)Dipartimento di Informatica, Università di Pisa, Italy, anna.bernasconi@unipi.it
\(^2\)Dipartimento di Informatica, Università degli Studi di Milano, Italy, valentina.ciriani@unimi.it
\(^3\)I.N.F.N., sezione di Milano, Italy, luca.frontini@mi.infn.it

NANOxCOMP
Emerging Technologies

New Integrated Circuits

- scaled CMOS is the main solution for digital IC
- reduce area and power consumption
- scaling will arrive to an end

investigate new emerging technologies

- develop tools for synthesis
- improve testability

post-CMOS Integrated Circuits
Overview

1. Preliminaries on **switching lattices** and on the **cellular fault model** (CFM).

2. Analysis of **cellular fault testability**.

3. Two methods for **improving the testability** of adjacent cellular faults in a lattice.

4. Experimental results.

5. Conclusions.
Switching Lattices are **two-dimensional** array of **four-terminal** switches. They are self-assembled devices fabricated with nano-fabrication techniques.

- When switches are ON all terminals are connected, when OFF all terminals are disconnected
- each switch is controlled by a boolean literal, 1 or 0
- the boolean function $f$ is the SOP of the literals along each path from **top** to **bottom**
- $f = x_1x_2x_3 + x_1x_2x_5x_6 + x_4x_5x_2x_3 + x_4x_5x_6$
For an easier representation the **crossbars** are converted to **lattices**:

- a), b): the 4-terminal switching network and the lattice describing
  \[ f = \overline{x_1} \overline{x_2} \overline{x_3} + x_1 x_2 + x_2 x_3 \]
- ‘checkerboard’ notation: darker and white sites represent **ON** and **OFF**
- c), d): the lattice with input \((1,1,0)\) and \((0,0,1)\)
Given a **Boolean function** $f$ and its dual function $f^D$:

1. find an **irredundant SOP** representation for $f$ and $f^D$:
   \[ SOP(f) = p_1 + p_2 + \ldots + p_s, \]
   \[ SOP(f^D) = q_1 + q_2 + \ldots + q_r; \]

2. form a $r \times s$ **switching lattice** and randomly assign each product $p_j$ of $SOP(f)$ to a column and each product $q_i$ of $SOP(f^D)$ to a row;

3. for all $1 \leq i \leq r$ and all $1 \leq j \leq s$, randomly **assign to the lattice cell** $c_{i,j}$ **one literal** that is shared by $q_i$ and $p_j$. 

\[
f = \overline{x_8}x_7\overline{x_6}x_3\overline{x_2}x_1 + \overline{x_8}x_7\overline{x_5}x_3\overline{x_2}x_1 + x_4x_3\overline{x_2}x_1 \]
• $f$ is synthesized from **top to bottom**
• the synthesis problem is formulated as a **satisfiability problem**, then the problem is solved with a SAT solver
• the synthesis method searches for better implementations starting from an upper bound size
• the synthesis loses the possibility to generate both $f$ and $f^D$

\[
f = \overline{x_8}x_7\overline{x_6}x_3\overline{x_2}x_1 + \overline{x_8}x_7\overline{x_5}x_3\overline{x_2}x_1 + x_4x_3\overline{x_2}x_1\]
A path is **unsatisfiable** if contains both a variable $x$ and $\overline{x}$.

The **product associated to a satisfiable path** is the conjunction of all literals of the path.

An **accepting path** for a minterm $v$ in a lattice is a satisfiable path whose associated product covers $v$.

A path is **prime** w.r.t. a literal $l_i$, if the product obtained removing $l_i$ from the path is not an implicant of the function.

The cell $c$ is **essential** if there exists at least a minterm $v$ in the on-set whose accepting paths always contain $c$. 

\[
f = x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5x_8x_9x_{10}x_{11} + \\
+ x_1\overline{x}_2\overline{x}_3\overline{x}_4\overline{x}_5x_8x_9x_{10}x_{11} + \\
+ x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5\overline{x}_7x_8 + x_1\overline{x}_2x_3x_4\overline{x}_7x_8 + \\
+ x_1\overline{x}_2x_3\overline{x}_4\overline{x}_5\overline{x}_7x_8 + x_1\overline{x}_2x_3x_4\overline{x}_7x_8\]
Cellular fault model in a Lattice

Let \( l_{i,j} \) be the literal in the cell \( c_{i,j} \):

- **R-ACF** Right Adjacent Cellular Fault is the cellular fault \((c_{i,j}, l_{i,j}, l_{i,j+1})\);
- **L-ACF** Left Adjacent Cellular Fault is the cellular fault \((c_{i,j}, l_{i,j}, l_{i,j-1})\);
- **T-ACF** Top Adjacent Cellular Fault is the cellular fault \((c_{i,j}, l_{i,j}, l_{i-1,j})\);
- **B-ACF** Bottom Adjacent Cellular Fault is the cellular fault \((c_{i,j}, l_{i,j}, l_{i+1,j})\).

\( T_L, T_R, T_B, \) and \( T_T \) are the number of testable cells with a L-ACF, R-ACF, B-ACF, and T-ACF.

\[
f = \overline{x}_8\overline{x}_7\overline{x}_6x_3\overline{x}_2x_1 + \overline{x}_8\overline{x}_7\overline{x}_5x_3\overline{x}_2x_1 + x_4x_3\overline{x}_2x_1
\]
Testability in cellular fault model

- **CF** \((c, l_c, l_f)\) in cell \(c\) with controlling literal \(l_c\) and faulty literal \(l_f\).
- The **test set** of the CF is the set \(T(l_c \leftarrow l_f)\) of all input vectors that give an uncorrected output on the faulty lattice.
- \(T(l_c \leftarrow l_f)\) are called **test vectors**.
- A fault is **testable if and only if** its test set is not empty.

**Example:**

\[ f = \overline{x}_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 + x_2 x_3. \]

- a) \(l_f = \overline{x}_1\), test vector \(\overline{x}_1 x_2 x_2\), the fault is testable;
- b) \(l_f = x_2\), no test vectors, the fault is not testable.
Proposition 1 and 2: testability of CFM

- A \( \text{CF}(c, l_c, l_f) \) in a lattice cell \( c \) with literal \( l_c \) is testable if and only if the \( \text{CF}(c, l_c, \bar{l}_c) \) is testable and the test set \( T_{(l_c \leftarrow \bar{l}_c)} \) contains at least one input vector where \( l_f \) and \( l_c \) assume different values.

\[
\begin{array}{ccc}
\bar{x}_1 & x_2 & x_2 \\
x_2 & x_1 & x_3 \\
x_3 & x_2 & x_2 \\
\end{array}
\quad
\begin{array}{ccc}
\bar{x}_1 & x_2 & \bar{x}_2 \\
x_2 & x_1 & x_3 \\
x_3 & x_2 & x_2 \\
\end{array}
\quad
\begin{array}{ccc}
\bar{x}_1 & x_2 & \bar{x}_1 \\
x_2 & x_1 & x_3 \\
x_3 & x_2 & x_2 \\
\end{array}
\]

Correct

a) \( \text{CF}(c, x_2, \bar{x}_2) \), b) \( \text{CF}(c, x_2, \bar{x}_1) \). a) is testable with test vector \( \bar{x}_1x_2x_3 \)

- A \( \text{CF}(c, l_c, \bar{l}_c) \) cannot be tested if for each path \( p \) through \( c \), the subpath \( p' = p \setminus \{c\} \) is unsatisfiable.

\[
\begin{array}{ccc}
x_1 & x_2 & x_2 \\
x_2 & \bar{x}_2 & x_3 \\
x_3 & x_2 & x_2 \\
\end{array}
\quad
\begin{array}{ccc}
x_1 & \bar{x}_3 & x_2 \\
x_2 & \bar{x}_2 & x_3 \\
x_3 & x_2 & x_2 \\
\end{array}
\quad
\begin{array}{ccc}
x_1 & \bar{x}_3 & x_2 \\
x_2 & \bar{x}_2 & x_3 \\
x_3 & x_2 & x_2 \\
\end{array}
\]

Correct

The blue subpath is unsatisfiable, so \( \text{CF}(c, x_2, x_3) \) is cannot be tested.
Testability of the CF\((c, l_c, \bar{l}_c)\)

- The **CF** \((c, l_c, \bar{l}_c)\) **can be tested on a path** \(p = p' \cup \{l_c\}\), where \(p'\) is satisfiable and contains an occurrence of \(\bar{l}_c\) if and only if **p is prime** with respect to \(l_c\).

  - In a) the CF\((c, x_2, \bar{x}_2)\) is testable, in b) the CF\((c, x_3, \bar{x}_3)\) is not testable.

- The **CF** \((c, l_c, \bar{l}_c)\) **can be tested on a path** \(p = p' \cup \{l_c\}\), where \(p'\) is satisfiable and contains an occurrence of \(l_c\) if and only if **c is essential**.

  - In a) the CF\((c, x_2, \bar{x}_2)\) is testable, in b) the CF\((c, x_3, \bar{x}_3)\) is not testable.

The cell \(c\) in a) is essential and the CF is testable, the cell \(c\) in b) is not essential and the CF not testable.
Testability of the CF\( (c, l_c, \overline{l}_c) \) and Theorem

- The CF \( (c, l_c, \overline{l}_c) \) can be tested on a path \( p = p' \cup \{l_c\} \), where \( p' \) is satisfiable and does not contain \( l_c \) or \( \overline{l}_c \) if and only if \( p \) is prime with respect to \( l_c \) or the cell \( c \) is essential.

\[
\begin{array}{ccc}
    & \text{TOP} & \\
    x_1 & x_2 & x_2 \\
    x_3 & x_3 & x_f \\
    x_2 & x_2 & x_2 \\
    \text{BOTTOM} & & \\
\end{array}
\]

Correct

\[
\begin{array}{ccc}
    & \text{TOP} & \\
    x_4 & x_2 & x_2 \\
    x_3 & x_3 & x_2 \\
    x_2 & x_2 & x_2 \\
    \text{BOTTOM} & & \\
\end{array}
\]
a)

\[
\begin{array}{ccc}
    & \text{TOP} & \\
    x_1 & x_2 & x_2 \\
    x_1 & x_1 & x_1 \\
    x_2 & x_2 & x_2 \\
    \text{BOTTOM} & & \\
\end{array}
\]
b)

The cell \( c \) in a) is essential and the CF is testable, the cell \( c \) in b) is not essential

**Theorem**

*For any literal \( l_f \) different form \( l_c \), the CF \( (c, l_c, l_f) \) in a lattice cell \( c \) with controlling literal \( l_c \) is testable if and only if \( T_{(l_c \leftarrow \overline{l}_c)} \cap B_{l_c \neq l_f} \neq \emptyset \).*

\( B_{l_c \neq l_f} \) is subset of the space \( \{0, 1\}^n \) where \( l_c \neq l_f \) assume different values

- With the test set for the fault \( (c, l_c, \overline{l}_c) \) it is possible to derive the test sets of all the other \( 2n - 2 \) cellular faults \( (c, l_c, l_f) \), where \( l_f \neq l_c \) and \( l_f \neq \overline{l}_c \)
Improving the testability in ACFM

- Improve testability of Adjacent Cellular Faults in lattices synthesized with Altun-Riedel (AR) synthesis method.
- AR defines many equivalent lattices \( r \times s \) for the same function \( f \).

To improve the lattice testability reducing the number of adjacent cell with the same literal we propose to:

1. **choose the best controlling literal** for each cell,
2. permute lattice **columns and rows**.

\[
f = x_1x_2 + x_1x_3 + x_2x_3
\]
Choose the best controlling literal

- **Heuristic algorithm** for choosing the controlling literal in $S_{i,j}$ for the cell $c_{i,j}$.
- The algorithm try to avoid to choice of controlling literals occurring in adjacent cells.

![Diagram](image)

ControllingLiterals (lattice $L$)

**INPUT:** a lattice $L$ $(r \times s)$ and, for each cell $c_{i,j}$, the set $S_{i,j}$ of its possible controlling literals

**OUTPUT:** a lattice $L'$ where each cell $c'_{i,j}$ contains exactly one controlling literal $l'_{i,j}$

for $i = 1$ to $r - 1$
  for $j = 1$ to $s - 1$
    $S = S_{i,j} \setminus (S_{i+1,j} \cup S_{i,j+1})$
    if ($S \neq \emptyset$) choose randomly $l'_{i,j} \in S$;
    else
      $S = S_{i,j} \setminus S_{i+1,j}$
      if ($S \neq \emptyset$) choose randomly $l'_{i,j} \in S$;
      else choose randomly $l'_{i,j} \in S_{i,j}$;
  
for $i = 1$ to $r - 1$ // last column
  $S = S_{i,s} \setminus S_{i+1,s}$
  if ($S \neq \emptyset$) choose randomly $l'_{i,s} \in S$;
  else choose randomly $l'_{i,s} \in S_{i,s}$;

for $j = 1$ to $s - 1$ // last row
  $S = S_{r,j} \setminus S_{r,j+1}$
  if ($S \neq \emptyset$) choose randomly $l'_{r,j} \in S$;
  else choose randomly $l'_{r,j} \in S_{r,j}$;

choose randomly $l'_{r,s} \in S_{r,s}$;
Permute lattice columns and rows

- Each product of the **SOP for** $f$ **is assigned to a column**.
- Each product of the **SOP for** $f^D$ **is assigned to a row**.
- Any **permutation** of the products in $SOP(f)$ and in $SOP(f^D)$ gives rise to a **correct lattice for** $f$.

- It is possible to **permute columns and rows** in order to **minimize the number of adjacent cells containing the same literal**.
- If two adjacent cells contain exactly the same literal, the corresponding ACF cannot be tested.

\begin{center}
\begin{tabular}{|c|c|c|}
  \hline
  \{x_1\} & \{x_1\} & \{x_2\} \\
  \hline
  \{x_1\} & \{x_1\} & \{x_3\} \\
  \hline
  \{x_2\} & \{x_3\} & \{x_2\} \\
  \hline
\end{tabular}
\end{center}

**not testable**

\begin{center}
\begin{tabular}{|c|c|c|}
  \hline
  \{x_1\} & \{x_2\} & \{x_1\} \\
  \hline
  \{x_1\} & \{x_3\} & \{x_1\} \\
  \hline
  \{x_2\} & \{x_2\} & \{x_3\} \\
  \hline
\end{tabular}
\end{center}

**testable**
A new version of Altun-Riedel algorithm

We propose a new version of Altun-Riedel algorithm in order to avoid some possible non-testable ACFs.

**Step 1:** find an irredundant, or a minimal, SOP representation for $f$ and $f^D$: $SOP(f) = p_1 + p_2 + \ldots p_s$ and $SOP(f^D) = q_1 + q_2 + \ldots q_r$;

**Step 2:** form a $r \times s$ switching lattice and assign each product $p_j$ $(1 \leq j \leq s)$ of $SOP(f)$ to a column and each product $q_i$ $(1 \leq i \leq r)$ of $SOP(f^D)$ to a row;

**Step 3:** for all $1 \leq i \leq r$ and all $1 \leq j \leq s$, assign to the switch on the lattice site $(i,j)$ one literal that is shared by $q_i$ and $p_j$ following the strategy described in the Algorithm;

**Step 4:** permute rows and columns in order to minimize the number of adjacent cells containing the same literal.
Experiments

- Benchmarks are taken from LGSynth93
- Each benchmark output is considered as a separate boolean function
- A total of 520 functions, we consider lattices with a number of variables lower than 6
- We compare the testability of ACFs for lattices obtained with Altun-Riedel (2012) and Gange-Søndergaard-Stuckey (2014) synthesis methods
- We evaluate the effect of the proposed lattice restructuring methods on the testability of lattices obtained with Altun-Riedel synthesis methods.
- We use a collection of Python scripts and a SAT solver to perform the Gange-Søndergaard-Stuckey synthesis.
- To compute the best permutation of rows and columns we use the linear optimizer GLPK (GNU Linear Programming Kit)

- The algorithm has been implemented in C
- The experiments have been run on a machine with 16 CPU @2.5 GHz, running Centos 7
We compare the number of testable cells for each between lattices synthesized with Altun-Riedel and Gange-Søndergaard-Stuckey methods.

The lattice synthesized with Gange-Søndergaard-Stuckey method contains a higher percentage of testable cells than Altun-Riedel in more than 70% of benchmarks.

<table>
<thead>
<tr>
<th>name</th>
<th>n</th>
<th>r</th>
<th>s</th>
<th>area</th>
<th>% $T_R$</th>
<th>% $T_L$</th>
<th>% $T_T$</th>
<th>% $T_B$</th>
<th>r</th>
<th>s</th>
<th>area</th>
<th>% $T_R$</th>
<th>% $T_L$</th>
<th>% $T_T$</th>
<th>% $T_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>add6(1)</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>36</td>
<td>53%</td>
<td>42%</td>
<td>67%</td>
<td>42%</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>87%</td>
<td>93%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>addm4(6)</td>
<td>5</td>
<td>10</td>
<td>11</td>
<td>110</td>
<td>55%</td>
<td>47%</td>
<td>24%</td>
<td>24%</td>
<td>6</td>
<td>4</td>
<td>24</td>
<td>100%</td>
<td>100%</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>bench(7)</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>24</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>3</td>
<td>5</td>
<td>15</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>ex5(35)</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>21</td>
<td>90%</td>
<td>86%</td>
<td>57%</td>
<td>57%</td>
<td>6</td>
<td>3</td>
<td>18</td>
<td>89%</td>
<td>89%</td>
<td>72%</td>
<td>67%</td>
</tr>
<tr>
<td>exp(13)</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>90%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>fout(1)</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>4</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>6</td>
<td>4</td>
<td>24</td>
<td>87%</td>
<td>92%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>fout(7)</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>80</td>
<td>64%</td>
<td>64%</td>
<td>27%</td>
<td>40%</td>
<td>6</td>
<td>4</td>
<td>24</td>
<td>92%</td>
<td>92%</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>fout(8)</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>90</td>
<td>61%</td>
<td>56%</td>
<td>27%</td>
<td>33%</td>
<td>6</td>
<td>4</td>
<td>24</td>
<td>96%</td>
<td>92%</td>
<td>87%</td>
<td>96%</td>
</tr>
<tr>
<td>risc(21)</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>80%</td>
<td>80%</td>
<td>90%</td>
<td>80%</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Z5xp1(5)</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>100</td>
<td>30%</td>
<td>29%</td>
<td>23%</td>
<td>28%</td>
<td>4</td>
<td>5</td>
<td>20</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>95%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Synthesis Method</th>
<th>Average area</th>
<th>$(T_R/\text{area})%$</th>
<th>$(T_L/\text{area})%$</th>
<th>$(T_T/\text{area})%$</th>
<th>$(T_B/\text{area})%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS</td>
<td>12</td>
<td>95.6%</td>
<td>95.7%</td>
<td>95.8%</td>
<td>95.4%</td>
</tr>
<tr>
<td>AR</td>
<td>27</td>
<td>69.1%</td>
<td>67.9%</td>
<td>68.1%</td>
<td>69.2%</td>
</tr>
</tbody>
</table>
Improving the testability of with Altum-Riedel method

Comparison between literal chosen randomly and with the proposed Algorithm

<table>
<thead>
<tr>
<th>name</th>
<th>n</th>
<th>col</th>
<th>row</th>
<th>area</th>
<th>Arbitrary</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>% $T_R$</td>
<td>% $T_L$</td>
</tr>
<tr>
<td>add6(2)</td>
<td>6</td>
<td>16</td>
<td>16</td>
<td>256</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>b12(0)</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>24</td>
<td>50%</td>
<td>37%</td>
</tr>
<tr>
<td>jbp(32)</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>m4(8)</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>m181(0)</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>24</td>
<td>50%</td>
<td>37%</td>
</tr>
<tr>
<td>mish(1)</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>30</td>
<td>60%</td>
<td>67%</td>
</tr>
<tr>
<td>shift(1)</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>62%</td>
<td>44%</td>
</tr>
</tbody>
</table>

Row and column permutations to minimize the number of adjacent cells that contain the same literal

<table>
<thead>
<tr>
<th>name</th>
<th>Col</th>
<th>Row</th>
<th>Area</th>
<th>n</th>
<th>ordered</th>
<th>randomly chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>% $T_R$</td>
<td>% $T_L$</td>
</tr>
<tr>
<td>add6(1)</td>
<td>6</td>
<td>6</td>
<td>36</td>
<td>4</td>
<td>69%</td>
<td>72%</td>
</tr>
<tr>
<td>alcom(2)</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>62%</td>
<td>62%</td>
</tr>
<tr>
<td>b12(0)</td>
<td>4</td>
<td>6</td>
<td>24</td>
<td>6</td>
<td>68%</td>
<td>62%</td>
</tr>
<tr>
<td>dc1(2)</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>4</td>
<td>75%</td>
<td>75%</td>
</tr>
<tr>
<td>inc(8)</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>67%</td>
<td>67%</td>
</tr>
<tr>
<td>mish(1)</td>
<td>5</td>
<td>6</td>
<td>30</td>
<td>6</td>
<td>77%</td>
<td>80%</td>
</tr>
<tr>
<td>radd(1)</td>
<td>6</td>
<td>6</td>
<td>36</td>
<td>4</td>
<td>69%</td>
<td>72%</td>
</tr>
</tbody>
</table>

Luca Frontini
Testability of Switching Lattices in the Cellular Fault Model
August 28, 2019
Conclusions

- we have extended the notion of cellular faults to switching lattices.
- we have proved that the testability of a general cellular fault is related to the testability of the inverted literal fault.
- We have exploited this result for simplifying the testability analysis of CFs.
- We proposed some techniques for improving the testability of a lattice for adjacent cellular fault without increasing lattice dimension.

Future works

- we will study different fault models for lattices.
- we will improve the testability of lattices synthesized with the method of Gange-Søndergaard-Stuckey.
Thank you!