ECE 586 – Fault Detection in Digital Circuits
Lecture 15 ATPG for SSFs I

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

March 4, 2015
Reading Assignment

- This lecture: 6.1, 6.2.1
- Next lecture: 6.2.1
Due: 5:00pm 03/25 Chicago time
Submit on Blackboard.
Outline

Automatic Test Pattern Generation

Fault-Oriented ATPG for SSFs in Combinational Circuits
Automatic Test Pattern Generation (ATPG)

- Generate test vectors and (good) output vectors for a circuit considering a fault universe.
  - Used in testing experiments to determine if any detectable fault exists by comparing good and possibly bad outputs.
  - We focus on combinational circuits and SSF model.

Considerations
- The cost of test generation (TG) – per design.
- The quality of the generated test.
- The cost of applying the test – per chip.

Random TG without considering a model of the circuit to be tested is conceptually simple.
- Cannot achieve high quality w/o incurring high testing cost.

We are interested in deterministic *automatic* TG that utilizes a structural model of the circuit.
- High quality, low testing cost.
- Though may need high TG cost.
Automatic Test Pattern Generation (ATPG)

- Generate test vectors and (good) output vectors for a circuit considering a fault universe.
  - Used in testing experiments to determine if any detectable fault exists by comparing good and possibly bad outputs.
  - We focus on combinational circuits and SSF model.

- Considerations
  - The cost of test generation (TG) – per design.
  - The quality of the generated test.
  - The cost of applying the test – per chip.

- Random TG without considering a model of the circuit to be tested is conceptually simple.
  - Cannot achieve high quality w/o incurring high testing cost.

- We are interested in deterministic automatic TG that utilizes a structural model of the circuit.
  - High quality, low testing cost.
  - Though may need high TG cost.
Automatic Test Pattern Generation (ATPG)

- Generate test vectors and (good) output vectors for a circuit considering a fault universe.
  - Used in testing experiments to determine if any detectable fault exists by comparing good and possibly bad outputs.
  - We focus on combinational circuits and SSF model.

Considerations

- The cost of test generation (TG) – per design.
- The quality of the generated test.
- The cost of applying the test – per chip.

Random TG without considering a model of the circuit to be tested is conceptually simple.

- Cannot achieve high quality w/o incurring high testing cost.

We are interested in deterministic automatic TG that utilizes a structural model of the circuit.

- High quality, low testing cost.
- Though may need high TG cost.
Automatic Test Pattern Generation (ATPG)

- Generate test vectors and (good) output vectors for a circuit considering a fault universe.
  - Used in testing experiments to determine if any detectable fault exists by comparing good and possibly bad outputs.
  - We focus on combinational circuits and SSF model.

- Considerations
  - The cost of test generation (TG) – per design.
  - The quality of the generated test.
  - The cost of applying the test – per chip.

- Random TG without considering a model of the circuit to be tested is conceptually simple.
  - Cannot achieve high quality w/o incurring high testing cost.

- We are interested in deterministic *automatic* TG that utilizes a structural model of the circuit.
  - High quality, low testing cost.
  - Though may need high TG cost.
Deterministic Test Generation System

Figure 6.1

(Abramovici et al., 1990)
Outline

Automatic Test Pattern Generation

Fault-Oriented ATPG for SSFs in Combinational Circuits
Algorithmic Idea: Line Justification

- SAT formulation: for a given fault $f$, find the test vector $t$ such that $OR(Z(t) \oplus Z_f(t)) = 1$.
  - The XOR operation $\oplus$ is bit-wise for a circuit with multiple POs.
  - In some sense, we will study how to solve this SAT problem.
- TG for a s-a-v fault $f$ at the line $l$,
  - Activate $f$ by partially setting $t$ to cause $l$ to have the value $\overline{v}$.
  - Propagate $f$ by completing $t$ to allow the value $\overline{v}$ on $l$ to cause one PO to be different from its good value.
- Line justification
  - Given some bits of $t$, a line $l$, and a value $v$.
  - Determine more bits of $t$ such that $l$ have the value $v$.
- Activating $f$ is solved directly as a line justification problem.
- Propagating $f$ can be decomposed into multiple line justification problems.
Algorithmic Idea: Line Justification

- SAT formulation: for a given fault $f$, find the test vector $t$ such that $OR(Z(t) \oplus Z_f(t)) = 1$.
  - The XOR operation $\oplus$ is bit-wise for a circuit with multiple POs.
  - In some sense, we will study how to solve this SAT problem.
- TG for a s-a-v fault $f$ at the line $l$,
  - Activate $f$ by partially setting $t$ to cause $l$ to have the value $\overline{v}$.
  - Propagate $f$ by completing $t$ to allow the value $\overline{v}$ on $l$ to cause one PO to be different from its good value.
- Line justification
  - Given some bits of $t$, a line $l$, and a value $v$.
  - Determine more bits of $t$ such that $l$ have the value $v$.
- Activating $f$ is solved directly as a line justification problem.
- Propagating $f$ can be decomposed into multiple line justification problems.
Algorithmic Idea: Line Justification

- SAT formulation: for a given fault $f$, find the test vector $t$ such that $OR(Z(t) \oplus Z_f(t)) = 1$.
  - The XOR operation $\oplus$ is bit-wise for a circuit with multiple POs.
  - In some sense, we will study how to solve this SAT problem.

- TG for a s-a-v fault $f$ at the line $l$,
  - Activate $f$ by partially setting $t$ to cause $l$ to have the value $\overline{v}$.
  - Propagate $f$ by completing $t$ to allow the value $\overline{v}$ on $l$ to cause one PO to be different from its good value.

- Line justification
  - Given some bits of $t$, a line $l$, and a value $v$.
  - Determine more bits of $t$ such that $l$ have the value $v$.

- Activating $f$ is solved directly as a line justification problem.
- Propagating $f$ can be decomposed into multiple line justification problems.
Algorithmic Idea: Line Justification

- SAT formulation: for a given fault $f$, find the test vector $t$ such that $\text{OR}(Z(t) \oplus Z_f(t)) = 1$.
  - The XOR operation $\oplus$ is bit-wise for a circuit with multiple POs.
  - In some sense, we will study how to solve this SAT problem.
- TG for a s-a-v fault $f$ at the line $l$,
  - Activate $f$ by partially setting $t$ to cause $l$ to have the value $\overline{v}$.
  - Propagate $f$ by completing $t$ to allow the value $\overline{v}$ on $l$ to cause one PO to be different from its good value.
- Line justification
  - Given some bits of $t$, a line $l$, and a value $v$.
  - Determine more bits of $t$ such that $l$ have the value $v$.
- Activating $f$ is solved directly as a line justification problem.
  - Propagating $f$ can be decomposed into multiple line justification problems.
Algorithmic Idea: Line Justification

- **SAT formulation:** for a given fault $f$, find the test vector $t$ such that $OR(Z(t) \oplus Z_f(t)) = 1$.
  - The XOR operation $\oplus$ is bit-wise for a circuit with multiple POs.
  - In some sense, we will study how to solve this SAT problem.
- **TG for a s-a-v fault $f$ at the line $l$,**
  - Activate $f$ by partially setting $t$ to cause $l$ to have the value $\overline{v}$.
  - Propagate $f$ by completing $t$ to allow the value $\overline{v}$ on $l$ to cause one PO to be different from its good value.
- **Line justification**
  - Given some bits of $t$, a line $l$, and a value $v$.
  - Determine more bits of $t$ such that $l$ have the value $v$.
- **Activating $f$** is solved directly as a line justification problem.
- **Propagating $f$** can be decomposed into multiple line justification problems.
To facilitate propagating both good and bad values, we introduce $D$ (good 1/bad 0) and $\bar{D}$ (good 0/bad 1).

To model bits in $t$ that are not decided yet, we introduce $x$ (unknown).

We only consider circuits made of AND/NAND/OR/NOR.

**Figure 6.2** Composite logic values and 5-valued operations

(Abramovici et al., 1990)
To facilitate propagating both good and bad values, we introduce $D$ (good 1/bad 0) and $\overline{D}$ (good 0/bad 1).

To model bits in $t$ that are not decided yet, we introduce $x$ (unknown).

We only consider circuits made of AND/NAND/OR/NOR.

---

**Figure 6.2** Composite logic values and 5-valued operations

(Abramovici et al., 1990)
Algorithm Outline: Test Generation in Fanout-Free Circuits

begin
set all values to $x$
$Justify(l, \overline{v})$
if $v = 0$ then $Propagate\ (l, D)$
else $Propagate\ (l, \overline{D})$
end

Figure 6.3  Test generation for the fault $l\ s-a-v$ in a fanout-free circuit

(Abramovici et al., 1990)

- Initially, all bits of $t$ are set to $x$.
- We don’t need to refer to good/bad circuits explicitly for error propagation by using $D$ (good 1) and $\overline{D}$ (good 0).
begin
set all values to $x$
Justify($l$, $\overline{v}$)
if $v = 0$ then Propagate ($l$, $D$)
else Propagate ($l$, $\overline{D}$)
end

Figure 6.3  Test generation for the fault
$l$ s-a-v in a fanout-free circuit

(Abramovici et al., 1990)

- Initially, all bits of $t$ are set to $x$.
- We don’t need to refer to good/bad circuits explicitly for error propagation by using $D$ (good 1) and $\overline{D}$ (good 0).
For fanout-free circuits, we assign 0/1 value to each bit of \( t \) at most once – no need to worry about conflicting assignments.

Depending on the value to justify, we either

- Request all inputs to have the non-controlling value, or
- Request an arbitrary input to have the controlling value.

Both requests are completed via recursion.

\[
\text{Justify} \ (l, \text{val}) \\
\begin{align*}
\text{begin} \\
\quad \text{set} \ l \ \text{to} \ \text{val} \\
\quad \text{if} \ l \ \text{is a PI then return} \\
\quad \quad /* \ l \ \text{is a gate (output) */} \\
\quad c = \text{controlling value of} \ l \\
\quad i = \text{inversion of} \ l \\
\quad \text{inval} = \text{val} \oplus i \\
\quad \text{if} \ (\text{inval} = \overline{c}) \\
\quad \quad \text{then for every input} \ j \ \text{of} \ l \\
\quad \quad \quad \text{Justify} \ (j, \ \text{inval}) \\
\quad \text{else} \\
\quad \quad \text{begin} \\
\quad \quad \quad \text{select one input} \ (j) \ \text{of} \ l \\
\quad \quad \quad \text{Justify} \ (j, \ \text{inval}) \\
\quad \text{end} \\
\text{end}
\]

\( \text{Figure 6.4} \) Line justification in a fanout-free circuit (Abramovici et al., 1990)
For fanout-free circuits, we assign 0/1 value to each bit of $t$ at most once – no need to worry about conflicting assignments.

Depending on the value to justify, we either

- Request all inputs to have the non-controlling value, or
- Request an arbitrary input to have the controlling value.

Both requests are completed via recursion.

---

### Justify ($l$, val)

begin

  set $l$ to val

  if $l$ is a PI then return

  /* $l$ is a gate (output) */

  $c = \text{controlling value of } l$

  $i = \text{inversion of } l$

  inval = $v\oplus i$

  if ($inval = c$)
    then for every input $j$ of $l$
      Justify ($j$, inval)
  else
    begin
      select one input ($j$) of $l$
      Justify ($j$, inval)
    end

end

Figure 6.4  Line justification in a fanout-free circuit (Abramovici et al., 1990)
Error Propagation in Fanout-Free Circuits

\begin{verbatim}
Propagate (l, err)
/* err is D or \overline{D} */
begin
  set l to err
  if l is PO then return
  k = the fanout of l
  c = controlling value of k
  i = inversion of k
  for every input j of k other than l
    Justify (j, \overline{c})
    Propagate (k, err \oplus i)
end
\end{verbatim}

\begin{itemize}
  \item Request all remaining inputs to have non-controlling values.
    \begin{itemize}
      \item By line justification.
    \end{itemize}
  \item Request to propagate the value at gate output.
    \begin{itemize}
      \item By recursion.
    \end{itemize}
\end{itemize}

Figure 6.5  Error propagation in a fanout-free circuit

(Abramovici et al., 1990)
Error Propagation in Fanout-Free Circuits

\[
\text{Propagate}\ (l, \text{err}) \\
/* \text{err is } D \text{ or } \overline{D} */ \\
\text{begin} \\
\text{set } l \text{ to err} \\
\text{if } l \text{ is PO then return} \\
\text{k = the fanout of } l \\
c = \text{controlling value of } k \\
i = \text{inversion of } k \\
\text{for every input } j \text{ of } k \text{ other than } l \\
\text{Justify}\ (j, \overline{c}) \\
\text{Propagate}\ (k, \text{err} \oplus i) \\
\text{end}
\]  

Figure 6.5  Error propagation in a fanout-free circuit

(Abramovici et al., 1990)

- Request all remaining inputs to have non-controlling values.
  - By line justification.
- Request to propagate the value at gate output.
  - By recursion.
Different choice at inputs of the gate driving $g$ will lead to different test vectors.

- e.g. 11x00
Conflict on \( a \) appears if we propagate \( \overline{D} \) to \( f_1 \).

- We could choose a different path to propagate.
  - In other cases, we may need to choose a different input to have the controlling value in line justification.
  - Need to try many different options since it is hard to know which one leads to conflicts.
The Case for Circuits with Fanouts

Conflict on $a$ appears if we propagate $\overline{D}$ to $f_1$.

We could choose a different path to propagate.

- In other cases, we may need to choose a different input to have the controlling value in line justification.
- Need to try many different options since it is hard to know which one leads to conflicts.
Resolve Conflicts by Choosing Different Input Values

If we choose $l = 1$ instead of $k = 1$ when justifying $q = 1$, then there will always be conflicts when justifying $r = 1$.

(Abramovici et al., 1990)
Summary

- TG in fanout-free circuits is simple because line justification will assign value to each PI at most once and there will be no conflicts.