ECE 586 – Fault Detection in Digital Circuits
Lecture 16 ATPG for SSFs II

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

March 9, 2015
Reading Assignment

- This lecture: 6.2.1
- Next lecture: 6.2.1
Outline

Fault-Oriented ATPG for SSFs in Combinational Circuits

Decision Tree
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.
Composite Logic Values and 5-Valued Operations

(a) \[ \begin{array}{c|cccccc} v/v_f & AND & 0 & 1 & D & \overline{D} & x \\ \hline 0/0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/1 & 1 & 0 & 1 & D & \overline{D} & x \\ 1/0 & D & 0 & D & D & 0 & x \\ 0/1 & \overline{D} & 0 & \overline{D} & 0 & \overline{D} & x \\ \hline x & 0 & x & x & x & x & x \end{array} \]

(b) \[ \begin{array}{c|cccccc} v/v_f & OR & 0 & 1 & D & \overline{D} & x \\ \hline 0/0 & 0 & 0 & 1 & D & \overline{D} & x \\ 1/1 & 1 & 1 & 1 & 1 & 1 & 1 \\ D & D & 1 & D & 1 & x \\ \overline{D} & \overline{D} & 1 & 1 & \overline{D} & x \\ \hline x & x & 1 & x & x & x & x \end{array} \]

(c) \[ \begin{array}{c|cccccc} v/v_f & OR & 0 & 1 & D & \overline{D} & x \\ \hline 0/0 & 0 & 0 & 1 & D & \overline{D} & x \\ 1/1 & 1 & 1 & 1 & 1 & 1 & 1 \\ D & D & 1 & D & 1 & x \\ \overline{D} & \overline{D} & 1 & 1 & \overline{D} & x \\ \hline x & x & 1 & x & x & x & x \end{array} \]

**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.
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.
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.
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)
Fault-Oriented ATPG for SSFs in Combinational Circuits

Decision Tree
TG as a Decision Process

- We solve TG by solving a series of sub-problems.
  - Justify and Propagate.
  - Smaller sub-problems can be generated during the process of solving larger sub-problems.

- A sub-problem may have a single solution – easy.
- A sub-problem may have many solutions.
  - Justify: when the output is controlled by any input.
  - Propagate: when the value is on a stem.
  - We need to make a choice.

- A sub-problem may have no solution.
  - i.e. the conflicts.
  - Justify: the line was assigned a different value.
  - Propagate: there is an controlling input.
  - What to do?
TG as a Decision Process

- We solve TG by solving a series of sub-problems.
  - Justify and Propagate.
  - Smaller sub-problems can be generated during the process of solving larger sub-problems.

- A sub-problem may have a single solution – easy.
- A sub-problem may have many solutions.
  - Justify: when the output is controlled by any input.
  - Propagate: when the value is on a stem.
  - We need to make a choice.

- A sub-problem may have no solution.
  - i.e. the conflicts.
  - Justify: the line was assigned a different value.
  - Propagate: there is an controlling input.
  - What to do?
TG as a Decision Process

- We solve TG by solving a series of sub-problems.
  - Justify and Propagate.
  - Smaller sub-problems can be generated during the process of solving larger sub-problems.
- A sub-problem may have a single solution – easy.
- A sub-problem may have many solutions.
  - Justify: when the output is controlled by any input.
  - Propagate: when the value is on a stem.
  - We need to make a choice.
- A sub-problem may have no solution.
  - i.e. the conflicts.
  - Justify: the line was assigned a different value.
  - Propagate: there is an controlling input.
  - What to do?
TG as a Decision Process

- We solve TG by solving a series of sub-problems.
  - Justify and Propagate.
  - Smaller sub-problems can be generated during the process of solving larger sub-problems.
- A sub-problem may have a single solution – easy.
- A sub-problem may have many solutions.
  - Justify: when the output is controlled by any input.
  - Propagate: when the value is on a stem.
  - We need to make a choice.
- A sub-problem may have no solution.
  - i.e. the conflicts.
  - Justify: the line was assigned a different value.
  - Propagate: there is an controlling input.
  - What to do?
Backtracking

- A conflict doesn’t mean the fault is not detectable.
  - We may have made a bad choice.
- Backtracking: we need to undo all the changes since the bad choice and try another solution.
  - If no solution is left, we can claim the fault is not detectable by any test vector.
- Need an algorithm.
  - Memorize the sub-problems we need to solve.
  - Memorize the sub-problems we have solved but alternative solutions exist.
  - Memorize all the changes so we can undo them.
- Note that we may need to make choices for multiple sub-problems before a conflict is found.
Backtracking

- A conflict doesn’t mean the fault is not detectable.
  - We may have made a bad choice.
- Backtracking: we need to undo all the changes since the bad choice and try another solution.
  - If no solution is left, we can claim the fault is not detectable by any test vector.
- Need an algorithm.
  - Memorize the sub-problems we need to solve.
  - Memorize the sub-problems we have solved but alternative solutions exist.
  - Memorize all the changes so we can undo them.
- Note that we may need to make choices for multiple sub-problems before a conflict is found.
Backtracking

- A conflict doesn’t mean the fault is not detectable.
  - We may have made a bad choice.
- Backtracking: we need to undo all the changes since the bad choice and try another solution.
  - If no solution is left, we can claim the fault is not detectable by any test vector.
- Need an algorithm.
  - Memorize the sub-problems we need to solve.
  - Memorize the sub-problems we have solved but alternative solutions exist.
  - Memorize all the changes so we can undo them.
- Note that we may need to make choices for multiple sub-problems before a conflict is found.
Backtracking

- A conflict doesn’t mean the fault is not detectable.
  - We may have made a bad choice.
- Backtracking: we need to *undo* all the changes since the bad choice and try another solution.
  - If no solution is left, we can claim the fault is not detectable by *any* test vector.
- Need an algorithm.
  - Memorize the sub-problems we need to solve.
  - Memorize the sub-problems we have solved but alternative solutions exist.
  - Memorize all the changes so we can undo them.
- Note that we may need to make choices for multiple sub-problems before a conflict is found.
Each node represents a sub-problem to be solved.
Each branch represents a solution.
Decision Tree Explained

Construct the decision tree.
- Nodes are created by choosing one sub-problem that is not solved yet.
- Branches are created by enumerating all possible solutions.
- Sub-problems that are not solved yet are attached to the branches.

We find a solution of the whole problem by traversing all the way down the tree.

If there is a conflict, we need to traverse up until we reach a node with a solution we haven’t tried yet.
- Then we traverse down from this solution.

The decision tree visualizes the progress of the algorithm.
- The tree may be generated on the fly.
- Storage requirement can be reduced by discarding the portion of the tree that is already traversed.
Construct the decision tree.

- Nodes are created by choosing one sub-problem that is not solved yet.
- Branches are created by enumerating all possible solutions.
- Sub-problems that are not solved yet are attached to the branches.

We find a solution of the whole problem by traversing all the way down the tree.

- If there is a conflict, we need to traverse up until we reach a node with a solution we haven’t tried yet.
  - Then we traverse down from this solution.
- The decision tree visualizes the progress of the algorithm.
  - The tree may be generated on the fly.
  - Storage requirement can be reduced by discarding the portion of the tree that is already traversed.
Construct the decision tree.
- Nodes are created by choosing one sub-problem that is not solved yet.
- Branches are created by enumerating all possible solutions.
- Sub-problems that are not solved yet are attached to the branches.

We find a solution of the whole problem by traversing all the way down the tree.

If there is a conflict, we need to traverse up until we reach a node with a solution we haven’t tried yet.
- Then we traverse down from this solution.

The decision tree visualizes the progress of the algorithm.
- The tree may be generated on the fly.
- Storage requirement can be reduced by discarding the portion of the tree that is already traversed.
Decision Tree Explained

- Construct the decision tree.
  - Nodes are created by choosing one sub-problem that is not solved yet.
  - Branches are created by enumerating all possible solutions.
  - Sub-problems that are not solved yet are attached to the branches.

- We find a solution of the whole problem by traversing all the way down the tree.

- If there is a conflict, we need to traverse up until we reach a node with a solution we haven’t tried yet.
  - Then we traverse down from this solution.

- The decision tree *visualizes* the progress of the algorithm.
  - The tree may be generated on the fly.
  - Storage requirement can be reduced by discarding the portion of the tree that is already traversed.
Example 1

- There are two sub-problems initially (at root).
  - Justify\((g,0)\), Propagate\((g,\overline{D})\)
  - We may solve Justify\((g,0)\) first.
    - And choose to solve all its sub-problems first.
  - Propagate\((g,\overline{D})\) has two solutions.
    - Propagate\((g_1,\overline{D})\), Propagate\((g_2,\overline{D})\).
  - Propagate\((g_1,\overline{D})\) has one solution Justify\((h_1,1)\).
    - Conflict on \(a\), backtrack to Propagate\((g,\overline{D})\).
  - ...

Figure 6.7

(Abramovici et al., 1990)
Example 1

There are two sub-problems initially (at root).
- Justify($g, 0$), Propagate($g, \overline{D}$)
- We may solve Justify($g, 0$) first.
  - And choose to solve all its sub-problems first.
- Propagate($g, \overline{D}$) has two solutions.
  - Propagate($g_1, \overline{D}$), Propagate($g_2, \overline{D}$).
- Propagate($g_1, \overline{D}$) has one solution Justify($h_1, 1$).
  - Conflict on $a$, backtrack to Propagate($g, \overline{D}$).
- ...
There are two sub-problems initially (at root).
  ▶ Justify\((g,0)\), Propagate\((g,\overline{D})\)
  ▶ We may solve Justify\((g,0)\) first.
    ▶ And choose to solve all its sub-problems first.
  ▶ Propagate\((g,\overline{D})\) has two solutions.
    ▶ Propagate\((g_1,\overline{D})\), Propagate\((g_2,\overline{D})\).
    ▶ Propagate\((g_1,\overline{D})\) has one solution Justify\((h_1,1)\).
      ▶ Conflict on \(a\), backtrack to Propagate\((g,\overline{D})\).
  ▶ ...

(Abrahmovici et al., 1990)
Example 1

There are two sub-problems initially (at root).
- Justify\((g,0)\), Propagate\((g,\overline{D})\)
- We may solve Justify\((g,0)\) first.
  - And choose to solve all its sub-problems first.
- Propagate\((g,\overline{D})\) has two solutions.
  - Propagate\((g_1,\overline{D})\), Propagate\((g_2,\overline{D})\).
- Propagate\((g_1,\overline{D})\) has one solution Justify\((h_1,1)\).
  - Conflict on \(a\), backtrack to Propagate\((g,\overline{D})\).

...
Example 1

There are two sub-problems initially (at root).
  - Justify($g,0$), Propagate($g,\overline{D}$)
  - We may solve Justify($g,0$) first.
    - And choose to solve all its sub-problems first.
  - Propagate($g,\overline{D}$) has two solutions.
    - Propagate($g_1,\overline{D}$), Propagate($g_2,\overline{D}$).
  - Propagate($g_1,\overline{D}$) has one solution Justify($h_1,1$).
    - Conflict on $a$, backtrack to Propagate($g,\overline{D}$).
  - ...

(Abramovici et al., 1990)
We could perform constant propagation after setting $a = 1$.

Then $h_1 = 0$ and $\text{Propagate}(g_1, \overline{D})$ has no solution.

In other words, we may save tree traversal by utilizing current solutions.

This may or may not save the overall running time.

Additional bookkeeping are necessary since the choice that leads to $a = 1$ could be bad.
Discussions

- We could perform constant propagation after setting $a = 1$.
  - Then $h_1 = 0$ and $\text{Propagate}(g_1, \overline{D})$ has no solution.
- In other words, we may save tree traversal by utilizing current solutions.
  - This may or may not save the overall running time.
  - Additional bookkeeping are necessary since the choice that leads to $a = 1$ could be bad.
Example 2

Figure 6.8

(Abramovici et al., 1990)
Discussions

- The size of the tree could be reduced if we allow to solve *multiple* sub-problems in a node.
  - Solve all sub-problems with a single solution (including the sub-problems generated by solving them).
  - Solve one sub-problem with multiple solutions.

- Save overall running time by reducing overheads to maintain sub-problems and to make function calls.
Discussions

- The size of the tree could be reduced if we allow to solve *multiple* sub-problems in a node.
  - Solve all sub-problems with a single solution (including the sub-problems generated by solving them).
  - Solve one sub-problem with multiple solutions.

- Save overall running time by reducing overheads to maintain sub-problems and to make function calls.
Example 3

(Fig. 6.12 (a), Abramovici et al., 1990)
The backtracking strategy can be visualized by a decision tree.