On-chip Dynamic Signal Sequence Slicing for Efficient Post-Silicon Debugging

Yeonbok Lee¹, Takeshi Matsumoto², Masahiro Fujita²,³

¹ Department of Electronic Engineering, University of Tokyo
² VLSI Design and Education Center, University of Tokyo
³ CREST, Japan Science and Technology Agency
Outline

- Introduction
- Proposed Method
- Experimental Results
- Summary and Future Works
Post-Silicon debugging

- **Post-Si Debugging?**
  - To find the root cause of an error detected by running fabricated chips

- **Post-Si debugging is getting mandatory**
  - Pre-silicon verification
    - Cannot be performed exhaustively
  - 71% of chip re-spins are due to design bugs [1]
    - Target of this work

- **Post-Si debugging is even more time-consuming & critical**

## Problems in post-Si debugging

- **Execution traces are very LONG**
  - Running speed of a chip is very fast (x 10^6~10^9 than simulation)
  - Amount of data to be analyzed is very large

- **Limited observability / Controllability**
  - Requires scan chain or on-chip trace buffer

| Scan-chain                | A snap-shot of the register values at each cycle can be obtained by one scanning-out
|                          | Available approach:
|                          | 1. Get internal register values at a timing by scanning out
|                          | 2. Re-simulate using the obtained values
|                          | Scanning-out once is time-consuming
|                          | “When to scan-out?” is an important problem

| On-chip trace memory      | Real-time signal values for multiple cycles can be obtained
|                          | Limited size for area constraint (8K~256K)
|                          | Available signal values are limited
Objective and contribution

Objective

- Propose a method to enhance the utility of scan-chain or on-chip trace memory
  - **Dynamic Signal Sequence Slicing Method**
    - Extracts signal values relevant to target error from the original LONG signal sequence

Contributions

- Save effort for analyzing irrelevant signal values
- Enhance the efficiency of post-silicon debugging strategy based on simulation using scanned-out values
  - Supporting the decision of when to scan-out
Outline

- Introduction
- **Proposed Method**
- Experimental Results
- Summary and Future Works
Basic assumptions

- Target design is described in FSMD (Finite State Machine with Data-path)
  - i.e., Controller part and data-path part are identical
  - FSMD = \langle Q, q_0, I, V, O, F, H \rangle

- An erroneous (output) value and its timing are known
  - E.g., by using assertion checker
**Terminologies**

- **Basic representation**
  - \([signal\_name](t)\) : the value of the signal named \(signal\_name\) at a clock cycle \(t\)

- **Definition of terminologies**
  - **Observing Signal Set \(S\)**
    - A set of signals that we want to observe.
    - \(S = \{si | si \in I \cup V \cup O\}\)
    - (Specified manually)
  - **Dynamic signal sequence slice of \(\chi(tn)\)**
    - Signal values having dependency to \(\chi(tn)\) from each signal value sequence of \(si\), i.e., \(\{si(0), \ldots, si(tn)\}\)
  - **Dynamic signal sequence slicing for \(\chi(tn)\)**
    - To obtain the dynamic signal sequence slice of \(\chi(tn)\)
FSMD (Finite State Machine)

Illustrative Example: when $S = 1$

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Ctrl State</th>
<th>in1</th>
<th>in2</th>
<th>in3</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>20</td>
<td>30</td>
<td>100</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>15</td>
<td>15</td>
<td>150</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>S3</td>
<td>10</td>
<td>20</td>
<td>100</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>S5</td>
<td>5</td>
<td>15</td>
<td>50</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>S6</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>-200</td>
</tr>
<tr>
<td>6</td>
<td>S1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>S2</td>
<td>5</td>
<td>0</td>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>S3</td>
<td>10</td>
<td>5</td>
<td>6</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>S4</td>
<td>15</td>
<td>10</td>
<td>8</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>S6</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>26</td>
</tr>
</tbody>
</table>

Expected results to be obtained
**d-tag (Dependency Tag)**

- **d-tag**
  - A new variable generate for each signal
  - A d-tag of signal $s$ is represented as $s_d$
  - $s_d(tn)$ represents whether a target signal value $o(tm)$ is dependent on $s(tn)$

**Definition of d-tag**

$$s_d(tn) = \begin{cases} 
1 & \text{A signal value } o(tm) \text{ is dependent on } s(tn) \\
0 & \text{Otherwise}
\end{cases}$$

- Obviously, $o_d(tm)$ for $o(tm)$ must be 1
- That is,
  - “To find the input signal values which the erroneous output value $o(tm)$ has dependency on”

  \[= \text{“To find the input values when the d-tag value of them become 1 w.r.t. } o_d(tm) \text{ specified as 1”} \]
The value of $s_d(t)$ of $s$:

- Can be inferred by the d-tag values of the signals which use the $s(t)$ in the next timing $t + 1$.

For example,

- For $S1$
  - $\text{reg} \leftarrow \text{in}
  \text{out} \leftarrow \text{out}$

- For $S2$
  - $\text{reg} \leftarrow \text{reg}
  \text{out} \leftarrow \text{reg}$

Generalization:

- $s_d(t) = \bigvee_q \left( \bigvee_{s_d' \leftarrow s \text{ at } q} x_d(t+1) \land (\text{state}(t+1) = q) \right)$

Thus, other d-tag values can be obtained by computing the equations \textit{backwardly} from a given $o_d(tm)$.
Example of d-tag computation

\[
\begin{align*}
\text{reg1}_d(t) &= (\text{out}_d(t+1) = 1) \land ((\text{state}(t+1) = S6) \\
&\quad \lor (\text{reg1}_d(t+1) = 1) \land ((\text{state}(t+1) = S5) \\
&\quad \lor (\text{reg1}_d(t+1) = 1) \land ((\text{state}(t+1) = S4) \\
&\quad \lor (\text{reg1}_d(t+1) = 1) \land ((\text{state}(t+1) = S3) \\
&\quad \lor (\text{reg1}_d(t+1) = 1) \land ((\text{state}(t+1) = S2) \\
&\quad \lor (\text{reg2}_d(t+1) = 1) \land ((\text{state}(t+1) = S1) \\
&\quad \lor (\text{reg2}_d(t+1) = 1) \land ((\text{state}(t+1) = S6) \\
&\quad \lor (\text{reg2}_d(t+1) = 1) \land ((\text{state}(t+1) = S5) \\
&\quad \lor (\text{reg2}_d(t+1) = 1) \land ((\text{state}(t+1) = S4) \\
\end{align*}
\]

FSMD (Finite State Machine)
Example of d-tag computation

\[ \text{reg1} \leq \text{in1} + \text{in2}; \]
\[ \text{reg2} \leq \text{in3} - 400; \]
\[ \text{reg1} \leq \text{reg1}; \]
\[ \text{reg2} \leq \text{reg1} + \text{reg2}; \]
\[ \text{status} \leq \text{reg1} > 10? \]
\[ \text{reg1} \leq \text{reg1} + \text{reg2}; \]
\[ \text{reg2} \leq \text{reg2}; \]
\[ \text{reg1} \leq \text{reg1} + 20; \]
\[ \text{reg2} \leq \text{reg2}; \]
\[ \text{out} = \text{reg1}; \]
\[ \text{in1}_d(t) = \text{reg1}_d(t+1) = 1 \land (\text{state}(t+1) = \text{S6}) \]
\[ \text{reg1}_d(t) = (\text{out}_d(t+1) = 1) \land ((\text{state}(t+1) = \text{S6}) \]
\[ \text{reg2}_d(t) = \text{reg2}_d(t+1) = 1 \land \text{reg1}_d(t+1) = 1 \land ((\text{state}(t+1) = \text{S5}) \]
\[ \text{in2}_d(t) = \text{reg1}_d(t+1) = 1 \land ((\text{state}(t+1) = \text{S5}) \]
\[ \text{reg2}_d(t) = \text{reg2}_d(t+1) = 1 \land ((\text{state}(t+1) = \text{S5}) \]
\[ \text{in3}_d(t) = \text{reg2}_d(t+1) = 1 \land ((\text{state}(t+1) = \text{S5}) \]

**FSMD (Finite State Machine)**
Dynamic signal sequence slicing

- **Basic procedure**
  1. Prepare equations for computing d-tags of signals in CUD (CUD: Circuit Under Debugging)
  2. In the erroneous execution,
     - Preserve the followings:
       - State transition history
       - Input signal value sequence
     - When the output signal becomes error:
       - Set its d-tag of to 1 (here, \( t = tm \))
  3. Compute the d-tags backwardly from \( t = tm \) toward \( t=0 \)
     - Return the input signals when the d-tags of them become 1 as result

- Can be realized either by SW or HW
An on-chip Implementation of Dynamic Signal Sequence Slicing

Input values

\[ \cdots \]
\[ i_m \]
\[ i_i \]
\[ \cdots \]

CUD

status

ctr

\[ t = t_n \]
\[ \cdots \]
\[ t = 2 \]
\[ t = 1 \]

Output values

\[ o_n \]
\[ o_1 \]
\[ \cdots \]

Trace memory

Input Values sequence

Control State history

Input values sequence

\[ \cdots \]

DSC

(Dynamic Sequence Slicing Circuit)

Computes d-tag values of each signal

Input value selector

Input signal values relevant to \( o_1 \) at \( t_n \)

\[ \cdots \]

\[ i_{1-d} \]
\[ i_{m-d} \]

( Assertion monitor )

d-tag values of output signals

d-tag values of input signals
Inputs and Outputs (when $S = I$) - for the on-chip implementation

- **Required items**
  - Data to be preserved in on-chip trace memory
    - Input signal value sequences
    - d-tag values of the output signals (1 when error)
    - Control state transition history
  - Additional Circuitry
    - For DSC circuit
      - FFs to preserve the d-tag values (0/1) of the signals in CUD
      - Combination circuits for computing the d-tags
    - A circuitry for extracting the input signal values when the d-tags of them become 1

- **The output results**
  - Input signal values that are relevant to the target error
An example of DSC

d-tag computation equations

\[
\begin{align*}
\text{reg}_d(t) &= ((\text{reg}_d(t+1)=1) \land (\text{state}(t+1)=S2)) \lor \\
&\quad ((\text{out}_d(t+1)=1) \land (\text{state}(t+1)=S2)) \\
\text{in}_d(t) &= (\text{reg}_d(t+1)=1) \land (\text{state}(t+1)=S1)
\end{align*}
\]
An execution example

Problem
Observed error: out(4) i.e.,\text{ out}_d(4) \leftarrow 1
\rightarrow To obtain the input signal values which out(4) is dependent on!

d-tag computation equations
\begin{align*}
\text{out}(t+1) &= 1 \quad \text{or} \quad \text{state}(t+1) = S2 \\
\text{out}_d(t+1) &= 1 \quad \text{and} \quad \text{state}(t+1) = S1
\end{align*}

Execution trace
\begin{array}{|c|c|c|c|c|c|}
\hline
 t & State & in & reg & out \\ \hline
 1 & S1 & 1 & X & X \\ \hline
 2 & S2 & 2 & 1 & 1 \\ \hline
 3 & S1 & 3 & 2 & 2 \\ \hline
 4 & S2 & 4 & 3 & 3 \\ \hline
\end{array}
An execution example

Problem
Observed error: out(4)
i.e., out_d(4) \leftarrow 1
→ To obtain the input signal values which out(4) is dependent on!

d-tag computation equations
\[
d(t+1) = 1 \land (state(t+1) = S2)) \lor
\]
\[
(state(t) = S1) \land (state(t+1) = S2)
\]

Executive trace
<table>
<thead>
<tr>
<th>t</th>
<th>State</th>
<th>in</th>
<th>reg</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Original Circuit

Buffer

Current timing \( t' = 2 \)

In_d = x
An execution example

reg <= in

Problem
Observed error: out(4)
i.e., out_d (4) ← 1
→ To obtain the input signal values which out(4) is dependent on!

<table>
<thead>
<tr>
<th>t</th>
<th>State</th>
<th>in</th>
<th>reg</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

d-tag computation equations

d(t+1)=1) ∧ (state(t+1) = S2) ∨
(t+1)=S2)

in_d(t) = (reg_d(t+1)=1) ∧ (state(t+1)=S1)

Original Circuit

Buffer

Execution trace

Current timing

t' = 3

Problem Observed error: out(4)
i.e., out_d (4) ← 1
→ To obtain the input signal values which out(4) is dependent on!

<table>
<thead>
<tr>
<th>t</th>
<th>State</th>
<th>in</th>
<th>reg</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

d-tag computation equations

d(t+1)=1) ∧ (state(t+1) = S2) ∨
(t+1)=S2)

in_d(t) = (reg_d(t+1)=1) ∧ (state(t+1)=S1)

DSC

Cmp (==)

S2

Cmp (==)

S1

reg_d

out_d=0

out_d=0

In_d= x
Problem

Observed error: \( \text{out}(4) \)
i.e., \( \text{out}_d(4) \leftarrow 1 \)
→ To obtain the input signal values which \( \text{out}(4) \) is dependent on!

**d-tag computation equations**

\[
d(t+1) = 1 \land (\text{state}(t+1) = S2)) \lor \\
(d(t+1) = S2)) \lor \\
(\text{state}(t+1) = S1)
\]

**Original Circuit**

**Buffer**

**Execution trace**

<table>
<thead>
<tr>
<th>( t )</th>
<th>State</th>
<th>( \text{in} )</th>
<th>reg</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

\( \text{out}_d(4) \leftarrow 1 \)

\( \text{reg}_d(4) \leftarrow 1 \)

\( \text{state}(4) \)

\( \text{Current timing } t' = 4 \)
An execution example

Problem

Observed error: out(4)

i.e., out_d(4) ← 1

→ To obtain the input signal values which out(4) is dependent on!

**Execution trace**

<table>
<thead>
<tr>
<th>t</th>
<th>State</th>
<th>in</th>
<th>reg</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**d-tag computation equations**

\[
\begin{align*}
\text{reg} & \leq \text{in} \\
\text{d}(t+1) &= 1 \land (\text{state}(t+1) = S2) \lor \\
(\text{out}_d(t+1) &= 1) \land (\text{state}(t+1) = S1) \\
\end{align*}
\]

**Recorded timing**

<table>
<thead>
<tr>
<th>t</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1</td>
</tr>
<tr>
<td>2</td>
<td>s2</td>
</tr>
<tr>
<td>3</td>
<td>s1</td>
</tr>
<tr>
<td>4</td>
<td>s2</td>
</tr>
</tbody>
</table>

**Buffer**

**Current timing**

\( t' = 5 \)
Problem

Observed error: out(4)
i.e., out_d(4) ← 1
→ To obtain the input signal values which out(4) is dependent on!

**d-tag computation equations**

\[
\begin{align*}
\text{reg}_d(t+1) &= (\text{state}(t+1) = S2) \lor (\text{state}(t+1) = S1) \\
\text{in}_d(t) &= (\text{reg}_d(t+1) = 1) \land (\text{state}(t+1) = S1)
\end{align*}
\]

**Execution trace**

<table>
<thead>
<tr>
<th>t</th>
<th>State</th>
<th>in</th>
<th>reg</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Current timing** \( t' = 6 \)
An execution example

**Problem**
Observed error: $\text{out}(4)$
i.e., $\text{out}_d(4) \leftarrow 1$
$\rightarrow$ To obtain the input signal values which $\text{out}(4)$ is dependent on!

<table>
<thead>
<tr>
<th>$t$</th>
<th>State</th>
<th>in</th>
<th>reg</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**d-tag computation equations**

$$d(t+1)=1 \land (\text{state}(t+1) = S2)) \lor$$

$$\lor \quad (\text{state}(t+1)=S1) \lor$$

$$\lor \quad (\text{state}(t+1)=S2))$$

**Original Circuit**

**Buffer**

**Current timing**
$t' = 7$

**Execution trace**

1. $t=1$
   - state = S1
   - $\text{in}_d = 0$
   - $\text{reg}_d(t) = 0$
   - $\text{out}(t) = X$

2. $t=2$
   - state = S2
   - $\text{in}_d = 0$
   - $\text{reg}_d(t) = 0$
   - $\text{out}(t) = 1$

3. $t=3$
   - state = S1
   - $\text{in}_d = 0$
   - $\text{reg}_d(t) = 0$
   - $\text{out}(t) = 1$

4. $t=4$
   - state = S2
   - $\text{in}_d = 0$
   - $\text{reg}_d(t) = 1$
   - $\text{out}(t) = 3$
Outline

- Introduction
- Proposed Method
- **Experimental Results**
- Summary and Future Works
Experiment

- Experiment
  - Implemented on-chip dynamic signal sequence slicing method on example designs
  - Evaluated the functionality and area overhead

- Example designs (RTL)
  - Simple: A design that performs simple arithmetic computation (362 lines)
  - Ellip: An elliptical filter design (1182 lines)
  - IDCT: 8x8 IDCT (1722 lines)

- Simulator
  - Cadence Verilog-XL

- RTL synthesizer
  - Synopsys Design Compiler
Experimental Results

<table>
<thead>
<tr>
<th></th>
<th># of input events</th>
<th># of Gates</th>
<th>Reduction rate</th>
<th># of Gates</th>
<th>DSC</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Extracted</td>
<td></td>
<td>Original</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>30,000</td>
<td>5,000</td>
<td></td>
<td>2758</td>
<td>156</td>
<td>5.65%</td>
</tr>
<tr>
<td>Ellip</td>
<td>10,000</td>
<td>769</td>
<td></td>
<td>16423</td>
<td>773</td>
<td>4.71%</td>
</tr>
<tr>
<td>IDCT</td>
<td>10,000</td>
<td>64</td>
<td></td>
<td>23672</td>
<td>462</td>
<td>1.95%</td>
</tr>
</tbody>
</table>

Furthermore, we implemented the designs on an FPGA board: Going to evaluate using “very long” input sequences.

Waveform of 8x8 IDCT design simulation and the slicing results:

D-tag of an output value under observing

Input values the d-tags of which become 1
Outline

- Introduction
- Proposed Method
- Experimental Results
- Summary and Future Works
Summary and Future Works

**Summary**

- Proposed a new variable d-tag representing the dynamic dependencies among signals in CUD
- Proposed dynamic signal sequence slicing method based on d-tag computation
- Proposed an on-chip implementation of dynamic signal sequence slicing method
- Evaluated functionality and area overhead using 3 designs
  - Relevant signal values to a specified signal value are extracted
  - Area overhead of the additional circuitry was 4% in average

**Future work**

- To examine various implementations
- To evaluate using “very long execution”