Enhanced Symbolic Simulation for Verification of Embedded Array Systems

Electrical & Computer Engineering Department, University of California, Santa Barbara
Tao Feng, Li-C Wang, K.T. Cheng
Enable symbolic simulation to verify large and complex memory system (MMU)

multiple array blocks and complex control logic
Outline

Symbolic simulation in the past

Approaches
- Integrate with an ATPG decision procedure
- Symbolic Encoding for input constraints
- Word-level Function Collapsing during simulation
- Use 2-tuple list (control, value)

Experimental Results

Conclusion
Symbolic simulation is effective for verifying individual embedded array block (STE)
Combine ATPG as justification decision procedure

- ATPG partitions the problem into a sequence of simpler sub-problems
- So that *symbolic encoding* can be applied more efficiently on each sub-problem.

- We assume that design-under-check has well-defined behavior model (antecedent=>consequent) convert the assertion checking into input constraints.
Backward Justification and Forward Simulation

ATPG + forward
Symbolic simulation

Circuit under Check

Constraint Netlist

symbolic Inputs (encoded constraints)

ATPG + Symbolic backward analysis (partition the problem)
Symbolic Initialization of Array cells

$((T0!=T) \lor (T1!=T) \lor (T2!=T) \lor (T3!=T)) \Rightarrow (\text{hit} = 0)$

Antecedent $\Rightarrow$ Consequent

Using $4w$ symbols in Antecedent to initialize array cells
Symbolic Encoding of Array cells

Symbolic index $p$ to choose one bit which is the complement bit in $T$

$T0[i] = \neg ti$ when $(p = i)$

$p$ is vector with $\lfloor \log w \rfloor$ bit, Using $4 \log(w)$ symbols to initialize array cells
False-negative effect for symbolic encoding

Word-level function Collapsing

hit0: \((A[0:10]=B[0:10])\&(\text{other conditions})\)
Word-level Collapsing during Simulation

Symbolic simulation

Circuit under Check

Assertion Model

all symbolic Inputs
no constraints

Word-level Collapse

Check results

0
Word-level function Collapse in Symbolic Simulation

1. Read the Gate level Netlist
2. Read Antecedent
3. Read Consequent
4. Event-Driven Simulate Circuit
5. Compare output BDD between circuit and assert
6. Level Circuit
7. Assign BDD variables
8. extract word-level function
9. build output BDD for assertion
10. collapse node with word-level function
11. build output BDD for circuit
Node Collapsing(hit0)

Original BDD without ordering
bdd_node_num: 110

Reordering (bottom down) variables
bdd_node_num: 81

Word-level function

After collapsing:
bdd_node_num: 5
2-tuple list representation for circuit node

Binary encoding:  \( L = \{(\text{express1, express2})\} \)
\{0, 1\} = 0;  \{1, 0\} = 1;  \{0, 0\} = \text{Unknown}

2-tuple list:  \( L : \{ (C_1, D_1), (C_2, D_2), \ldots, (C_n, D_n) \} \)

Signal L’s value is \( D_i \) when \( C_i \) is true.

Split one complex bool expression into some simpler expressions
2-tuple list representation for circuit node

With 2-tuple list
- Simulate the CAM read/write/translate mode in one iteration
  Output: { (read control mode, value)
            (write control mode, value)
            (translate control mode, value) }
2-tuple list Construction

\[
\begin{align*}
A & \Rightarrow ((Ca_1, Da_1)) \quad O \Rightarrow ((Ca_i & Cb_1, \ Da_i \& \ Db_i), \\
B & \Rightarrow ((Cb_1, Db_1)) \quad \quad (Ca_i \& \ Cb_1 \& \ Da_i) | (Ca_i \& \ Cb_1 \& \ Db_i, \ 0))
\end{align*}
\]

\[
\begin{align*}
A & \Rightarrow ((Ca_1, Da_1)) \quad O \Rightarrow ((Ca_i & Cb_1, \ Da_i | \ Db_i), \\
B & \Rightarrow ((Cb_1, Db_1)) \quad (Ca_i \& \ Cb_1 \& \ Da_i) | Ca_i \& \ Cb_1 \& \ Db_i, \ 1))
\end{align*}
\]

\[
\begin{align*}
A & \Rightarrow ((Ca_1, Da_1)) \quad O \Rightarrow (Ca_i, \ Da_i)
\end{align*}
\]

\[
\begin{align*}
A & \Rightarrow ((Ca_1, Da_i)) \quad O \Rightarrow ((Ca_i & Da_i & Ce_i, \ De_i), \\
B & \Rightarrow ((Cb_1, Db_i)) \quad (Ca_i \& \ Da_i \& \ Ce_i, \ Db_i))
\end{align*}
\]

2-tuple list Merge

\[
\{(C0, D0), (C1, D0)\} \Rightarrow \{(C0|C1, D0)\}
\]
If some errors in the circuit ….

Encoding and word-level collapsing methods depends on the golden model of CAM. Error in the circuit will violate this assumption, and influence the performance of the methods.
**Experimental Result**

<table>
<thead>
<tr>
<th>Methods</th>
<th>SS-wout</th>
<th>SS-w</th>
<th>SS-encode*</th>
<th>SS2-wout</th>
<th>SS2-cop-wout</th>
<th>SS2-cop-w</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time(sec)</td>
<td>Abort</td>
<td>1038.2</td>
<td>11.7</td>
<td>224</td>
<td>21</td>
<td>3</td>
</tr>
</tbody>
</table>

SS: Binary encoding simulator (Voss)
SS2: 2-tuple list simulator
SS2-cop: 2-tuple list simulator with node collapse
-wout: without initial good ordering given manually
-w: with initial good ordering given manually

Except SS-encode*, other methods can verify MMU read/write/translate function in one simulation.
Various kind of errors are injected into the circuit, how do they influence the performance of the methods?

<table>
<thead>
<tr>
<th></th>
<th>SS-w</th>
<th>SS-encode</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time(sec)</td>
<td>OBDD size</td>
<td>Time (sec)</td>
<td>OBDD size</td>
</tr>
<tr>
<td>No Error</td>
<td>95.6</td>
<td>31902</td>
<td>11.7</td>
<td>8416</td>
</tr>
<tr>
<td>Extra Inv</td>
<td>93.7</td>
<td>31417</td>
<td>9.2</td>
<td>6551</td>
</tr>
<tr>
<td>Gate Sub</td>
<td>256.7</td>
<td>33902</td>
<td>11.7</td>
<td>8209</td>
</tr>
<tr>
<td>Wrong Wires</td>
<td>268.4</td>
<td>79391</td>
<td>48.8</td>
<td>8944</td>
</tr>
<tr>
<td>Hier Extra Inv</td>
<td>94.8</td>
<td>23021</td>
<td>Can not handle</td>
<td></td>
</tr>
</tbody>
</table>
### Experimental Result

<table>
<thead>
<tr>
<th>Error Type</th>
<th>SS2-wout Time</th>
<th>SS2-cop-wout Time</th>
<th>new var</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>224</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>Extra Inv</td>
<td>223</td>
<td>44</td>
<td>8</td>
</tr>
<tr>
<td>Gate Sub</td>
<td>230</td>
<td>44</td>
<td>12</td>
</tr>
<tr>
<td>Wrong Wires</td>
<td>262</td>
<td>42</td>
<td>8</td>
</tr>
<tr>
<td>Hier Extra Inv</td>
<td>Can not</td>
<td>Can not</td>
<td>0</td>
</tr>
</tbody>
</table>

Error will cause the simulator fail to collapse some word-level functions.
For the single error, the collapse method still works well.
For the worst case with hierarchy errors, it may cause the simulator incapable to verify the whole circuit.
**Summary**

- ATPG as a decision procedure to partition a given assertion into simpler sub-assertions
- Symbolic Encoding for input constraints
- Word-level Function Collapsing during simulation
- Use 2-tuple list (control, value)
- Study the methods’ stability with error injected
- This combined strategy allows us to verify whole MMU