Automatic Generation of Schedulings for Improving the Test Coverage of Systems-on-a-Chip

Claude Helmsitter and Florence Maraninchi and Laurent Maillet-Contoz and Matthieu Moy

Verimag & ST Microelectronics

FMCAD'06
Context: Transaction Level Models

- **Fastest**: Early simulation of the embedded software. Golden model for RTL validation.
- **Most Precise**: System-on-a-Chip (SoC) synthesis.

Transaction Level Models (TLM) are used for early simulation of the embedded software, providing a Golden model for RTL validation. System-on-a-Chip (SoC) synthesis is the most precise level, focusing on detailed design and verification processes.
Elaboration phase, non-preemptive scheduling, simulated time.
Example of Scheduling Dependencies

```cpp
void top::P() {
    wait(e);
    wait(20);
    if (x) cout << "Ok\n";
    else cout << "Ko\n";
}

void top::Q() {
    e.notify();
    x = 0;
    wait(20);
    x = 1;
}

• 3 possible schedulings: (TE=Time Elapse)
  • P_1;Q_1;P_2;[TE];Q_2;P_3: Ok
    default OSCI scheduler choice, if P declared before Q and if ...
  • P_1;Q_1;P_2;[TE];P_3;Q_2: Ko
  • Q_1;P_1;[TE];Q_2: “dead-lock”
```
The Coverage Problem

- Even if data is fixed
  - The SystemC LRM allows many schedulings
  - Some implementations are not deterministic

- For the validation of SoC models:
  - 1 execution => very poor coverage
  - Random schedulings => uncertain coverage, lots of useless executions
  - Test with all possible schedulings => unrealistic

- Our goal: **test only executions that may lead to different final states**
Outline

- TLM, SystemC and the Coverage Problem
- Principle of the Technique Applied
- Implementation and Results
- Conclusion
Principle of the Approach

Data is fixed; we generate schedulings

Use of Dynamic Partial Order Reductions (presented by C. Flanagan, P. Godefroid at POPL'05)
Cyclic Generation

Test directives for new executions

Program.exe + data

(0..n)

Checker

Execution trace

Checked trace (~ partial order)
Checker: Observing Traces

Goal:
Guess if transitions are dependent by observation of their behavior

$p$: wait($e$) \quad $q$: $e$.notify()
Checker: Action Dependencies

- Independent $\iff$ order is irrelevant
- Dependency cases for SystemC:
  - Variables (or memory locations):
    - Two write ($T[12] = 1$ and $T[12] = 2$)
    - One write and one read ($x = 1$ and $f(x)$)
  - Events:
    - One notify and one wait
    - In some cases: two notify (consequences on the computed partial order)
Checker: Dynamic Dependency Graph

Execution Trace:

- \( p_1 \): wait(e)
- \( q_1 \): notify(e), modify(x)
- \( p_2 \): enabled by \( q_1 \)
- \( q_2 \): modify(x)
- \( p_3 \): read(x)

[Time Elapse] Dynamic Dependency Graph:

- Green arrows: dependent but not permutable
- Red arrows: dependent and permutable
Checker: Scheduling Constraint

Generation of 1 new test directive for each red arrows

$\textbf{p}_i < \textbf{q}_j$: $i$-th execution of process $\textbf{p}$ before $j$-th execution of process $\textbf{q}$
Cyclic Generation with Scheduling Constraints

\{Q_1 > P_1, P_3 > Q_2\} \rightarrow \{Q_1 > P_1\}

Program.exe

Checker

One new constraint set

Set of inherited constraints (from previous checking)

TRACE
Transition|Actions
\begin{align*}
P_1 & \quad \text{wait}(e) \\
Q_1 & \quad \text{notify}(e), \text{modify}(x) \\
P_2 & \quad \text{enabled by } Q_1 \\
P_3 & \quad \text{read}(x) \\
Q_2 & \quad \text{modify}(x)
\end{align*}

Checker

{Q_1 > P_1}
Property Guaranteed by this Method

- **A**: Set of all possible executions (for one data)
- **G**: Set of generated executions (for the same data)

**Property**: For all \( a \) in \( A \), there exists \( g \) in \( G \) that differs only by the order of independent transitions.

**Consequences on coverage:**
- Full code accessibility for each process
- All Dead-locks found
Proof Hint: Constraint Trees

Define a function $f$ from $A$ to $G$.

- Define a function $f$ from $A$ to $G$.
- $a$ and $f(a)$ differ only by the order of independent transitions.
Outline

- TLM, SystemC and the Coverage Problem
- Principle of the Technique Applied
- Implementation and Results
- Conclusion & Demo
The Tool Chain

SystemC model

Pinapa analyzer

Instrumented model

patched SystemC kernel

new constraints

raw trace

checked trace

checker
Industrial Case Study: LCMPEG

- Part of a Set-Top Box, from STM
- 5 components, runs of 150 transitions, with long sections of sequential code (~50k lines)
- 32 generated schedulings (\(=\text{card}(G)\)), 13 sec
- 9.5 sec for the 32 simulations / 3.5 sec of overhead (time spent in checker)
- At least \(2^{40}\) possible schedulings (\(=\text{card}(A)\))
Extension: Validation of SoC models in the presence of loose timings

- New instruction: \texttt{lwait(42±12)} for representing unprecise timings
  - basic implementation: random in [30,54]
  - better: DPOR + Linear Programming
- Results: LCMPEG with delays ± 20% => 3584 simulations, 35 min 11 sec.
- Presented at FMICS'06
Conclusion

- Already works on medium-sized industrial case studies
- Should work on larger case studies with some improvements
- Well adapted to abstract TLM models which are asynchronous
- Light tool: no explicit extraction of an abstract formal model, no state comparison, ...
Further Works

- Validation of SoC models in the presence of loose timings (presented at FMICS'06)
- Possible optimizations:
  - higher level synchronization mechanisms (persistent events)
  - remove useless branches of constraint trees
  - parallelization of the prototype
- Parallelization of the SystemC engine (based on dependency analysis too)
Thank you for your attention.
### Persistent Events

| Process A: v = 1; e.notify(); |
| Process B: if (!v) wait(e); v = 0; |

- **Consequence:** useless simulations
- **Solution:**
  - new class `pevent` with methods `wait`, `notify` and `reset`
  - extending dependency analysis
- **Result:** from 128 to 32 generated schedulings for the LCMPEG