Towards Proving TLM Properties with Local Variables

Hoang M. Le
Daniel Große*
Rolf Drechsler

University of Bremen, Germany
grosse@informatik.uni-bremen.de
Outline

- Motivation
- SystemC
- TLM Property Checking
- TLM Properties with Local Variables
- Experimental Results
- Conclusions
Motivation

- ESL design
- Correctness of the initial TLM model
  - Deadlock-freedom, local assertions, causality between transactions and events
  - Data integrity?
What is SystemC?

• C++ class library
  concurrency   clocks
  hierarchy    HW data types  ...

• Simulation kernel
  - Non-preemptive execution semantic
  - Event-driven

• Transaction Level Modeling (TLM)
  - Separation of communication and functionality
  - Communication by method calls: `write(data, addr)`
  - No detailed protocols, no clocks, ...
Running Example (1)

```c
void write(char c_in) {
    while (num_elmts == max) {
        wait(read_event);
        data[(first + num_elmts) % max] = c_in;
        num_elmts++;
        write_event.notify();
    }
}

void read(char &c_out) {
    while (num_elmts == 0) {
        wait(write_event);
        c_out = data[first];
        --num_elmts;
        // first = (first + 1) % max; bug
        read_event.notify();
    }
}
```
Running Example (2) – An Execution

```
Write A → Write B → Write C → Write D → Write E

wait(read_event);
FIFO = ABCDE

Read A → Read A → Read A → Read A → Read A

wait(write_event);
Empty FIFO

...```

Outline

- Motivation
- SystemC
- **TLM Property Checking**
  - Model generation
  - Property language & monitor generation
  - BMC-based formulation
- TLM Properties with Local Variables
- Experimental results
- Conclusions
Model Generation (SystemC to C)

**Step 1: Basic transformation**
- Identify elaborated structure
- Translate OO features to C

**Step 2: Scheduler generation**
- Scheduler loops
- Non-deterministic process selection
- Code for process execution

**Step 3: Events & Context Switches Handling**
- Test/set primitive variables
- Break/continuation of process with jumps
Properties & Monitors (1)

- **PSL based** (see Tabakov et al. FMCAD08)
- **Primitives:**
  - Variables
  - Return value and parameters of functions
  - Begin & end of transaction
  - Event notification

- **“Time”:**
  - Sample at all system events
  - Change of resolution by clock expressions
Properties & Monitors (2)

- **TLM Properties**
  - Notification of events, begin & end of transactions and *order of occurrence*

  ```
  default clock = write_event.notified || read_event.notified;
  always (write_event.notified -> next_e[1:10] read_event.notified)
  ```

- **Monitors**
  - Translate property to FSM
  - Embedded into C model as C++ assertions
BMC & Induction

- Transition relation
  - State $s = \text{current values of variables}$
  - $T$ defined by outermost loop of scheduler

- Formulation

\[
allSafe(s_{[0..n]}) = \bigwedge_{0 \leq i < n} safe(s_i, s_{i+1})
\]

\[
\exists s_0 \ldots s_k. (I(s_0) \land \text{path}(s_{[0..k]}) \land \neg allSafe(s_{[0..k]}))
\]

$k = 0, 1, 2 \ldots$ (number of unwound main loops)

- Induction (see Große et al. MEMOCODE10)
Outline

• Motivation
• SystemC
• TLM Property Checking
• TLM Properties with Local Variables
  – Syntax & Semantics
  – Monitoring Logic
  – Optimization
• Experimental results
• Conclusions
Syntax & Semantics

• Syntax
  – Local variable in TLM property by
    ```
    var x = exp
    ```
  – x refers to actual value
    ```
    default clock = write_event.notified || read_event.notified;
    always ((write_event.notified, var x = c_in)
      -> next_e[1:10] (read_event.notified && c_out == x))
    ```

• Semantics
  – Based on FSM
  – Tokens for overlapping evaluations
  – Token holds local variables
One Evaluation

\[ r \&\& c_{\text{out}} == v \]

\[ w, v = c_{\text{in}} \]

\[ \text{cond} = w || (r \&\& v != c_{\text{out}}) \]
Overlapping Evaluations

\[ r \land \land c_{\text{out}} = v \]

\[ w, v = c_{\text{in}} \]

\[ \text{cond} \]

\[ \text{Write } A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \]

\[ \text{Read } A \rightarrow A \rightarrow A \rightarrow A \rightarrow A \]

\[ \text{cond} = w \lor (r \land v \neq c_{\text{out}}) \]
Monitoring Logic

- **Embedding allows Formal PC**
- **Token**
  - $n$ values of $n$ local variables
  - 1 value for FSM position
  - Statically allocated
- **Number of tokens**
  - Upper-bound = bound of the property (syntactically derivable)
  - Exact number $\implies$ incremental PC
Optimization

- **Avoid overlapping evaluations**
  - Only 1 token necessary
  - Start evaluation from non-deterministic state $\Rightarrow$ all possible evaluations covered implicitly
  - Unwinding depth sufficient? $\Rightarrow$ counter for passed sampling points
Experiments

- CBMC v4.0, AMD 3.4 GHz, 8GB RAM, Linux
- FIFO design

<table>
<thead>
<tr>
<th>FIFO Size</th>
<th>IP</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>max = 5</td>
<td>47.90s</td>
<td>22.25s</td>
</tr>
<tr>
<td>max = 10</td>
<td>877.43s</td>
<td>220.69s</td>
</tr>
</tbody>
</table>
Conclusions

• Local variables support for TLM-PC
  – Data integrity formally specified and proven
  – Optimization by using non-determinism

• Future work:
  – Automation
  – Extension for other property classes