SystemC-based ESL Verification Flow
Integrating Property Checking and Automatic Debugging

Hoang M. Le¹
Daniel Große¹
Rolf Drechsler¹,²

¹University of Bremen, Germany
²DFKI Bremen, Germany
{hle, grosse, drechsle}@informatik.uni-bremen.de
Outline

• Motivation
• SystemC
• Overall Verification Flow
• TLM Property Checking
• Automatic TLM Debugging
• Conclusions
Motivation

- **ESL design**
- **Correctness of the initial ESL model**
  - Does the design satisfy the specification?
  - Which part of the design is responsible for the error?
**Running Example (1)**

```
**class** sender : **public** sc_module {
    **private**
    static **void** main() {
        while (**true**) {
            done_receiving.notify();
        }
    }
};

**class** receiver : **public**
receiver_if, **public** sc_module {
    **private**
    **void** receive(...) {
        data = x;
        done_receiving.notify();
        wait(done_processing);
    }
};
```

```cpp
**void** main() {
    while (**true**) {
        port->receive(v);
        v++;
    }
}
```
Running Example (2)

Receiver

```c
void main() {
    while (true) {
        wait(done_receiving);
        if (data != 0) port->add(data);
        done_processing.notify();
    }
}
```

Sender

```c
void main() {
    while (true) {
        wait(done_receiving);
        if (data != 0) port->add(data);  
        done_processing.notify();
        receive(1);
        data = 1;
        done_receiving.notify();
        wait(done_processing);
        add(1);
        done_processing.notify();
    }
}
```

Receiver

```c
void receive(uint x) {
    data = x;
    done_receiving.notify();
    wait(done_processing);
}
```

```
```

sender main()

wait(done_receiving);

receive(0);
data = 0;
done_receiving.notify();
wait(done_processing);

receiver main()

done_processing.notify();

add(1);
done_processing.notify();

```
Overall Verification Flow

- TLM Property in PSL
- SystemC TLM design
- Possible Fault Locations
- Automatic TLM Debugging
- Error Trace

Flow:
- TLM Property Checking
  - ok: Done
  - failed

Actions:
- fix bug
TLM Property Checking

1. SystemC TLM design
2. TLM Property in PSL
3. Generation of C model
4. Generation of monitoring logic
5. BMC/Induction on final C model
6. Trace found?
   - next property if no
   - debugging if yes
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

- Simple safety
  - `always` \((\text{data} \neq 0)\)
- Notification of events, begin & end of transactions and *order of occurrence*
  - `always` \((\text{done	extunderscore receiving}:\text{notified} -> \text{next} ((\text{data} \neq 0) -> \text{sub}:entry))\)
- Local variables
  - `always` \((\text{write} \text{event}:\text{notified}, \text{var} x = \text{c\_in}) -> \text{next}_e[1:\text{max}] (\text{read} \text{event}:\text{notified} \&\& \text{c\_out} == x) \)
Properties & Monitors (3)

• Monitors
  - Translate property -> FSM
  - Tokens for local variables
  - Embedded into the model as C++ assertions
BMC

• Transition relation
  - State s = current values of variables
  - $T$ defined by outermost loop of scheduler

• Formulation

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

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

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

• Induction (see Große et al. MEMOCODE10)
Back to the example

- **Contradiction**
  - **always** (done_receiving: notified -> **next** ((data != 0) -> sub: entry))
Automatic Debugging

- **Basic Idea**
  - Possibly Faulty Part (PFP)
    - TLM-related constructs
    - Prone to errors
  - Non-reproducibility of error trace
    - Bug-fix eliminates error trace
    - Transformation of a PFP eliminates error trace -> PFP is possible fault location
Debugging Flow

SystemC TLM design → Transformation of PFP → PFP

Error trace → Error trace injection

TLM Property in PSL → BMC on final C model

Add PFP to set of possible fault locations

Trace found? (yes) → Next PFP (no)

Generation of C model → Constraining the execution
Fault Model (1)

- **Transaction**
  - **Fault:** Initiation of an incorrect transaction (e.g. blocking vs. non-blocking)
  - **Correction:** replace with another transaction

- **Transaction data**
  - **Fault:** Call a transaction with an incorrect argument (e.g. wrong bus adress)
  - **Correction:** allow the argument to be non-deterministic
Fault Model (2)

- **Concurrent function**
  - **Fault:** Incorrect use of wait/notify (e.g. notify too early)
  - **Correction:** exchange timed/untimed, non-deterministic time argument

- **Event**
  - **Fault:** wait for/notify an incorrect event
  - **Correction:** replace with another event
Non-reproducibility check

• **Error trace injection**
  – Fixed design inputs
  – Fixed schedule sequence (unroll main loop)

• **Constraining the execution**
  – Only compliant traces
  – Replacement of assert with assume

• **BMC on final C model**
  – Compliant trace = Counter-example
  – Presence -> error trace not reproducible
Symbolic Encoding

• **diag Variable**
  - Non-deterministic
  - Value – Transformation mapping

• **Controlled transformations**
  - Replace original PFP with a block of if-statements

```plaintext
diag = nondet();
...
if (diag == 1) done_processing.notify();
else
  if (diag == 2) one_receiving.notify(nondet());
  else done_receiving.notify(); // original
...
if (data != 0) {
  if (diag == 7) port->sub(data);
  else
    if (diag == 8) port->add(nondet());
    else port->add(data); // original
}
Summary

- Integrated Verification Flow
- Efficient verification of high-level properties
- Accelerated debugging by automatic fault localization

Increased verification productivity
SystemC-based ESL Verification Flow Integrating Property Checking and Automatic Debugging

Hoang M. Le¹
Daniel Große¹
Rolf Drechsler¹,²

¹University of Bremen, Germany
²DFKI Bremen, Germany
{hle, grosse, drechsle}@informatik.uni-bremen.de