Testing Resource Isolation for SoC Architectures

Philippe Ledent  Radu Mateescu
Wendelin Serwe
INRIA – Laboratoire d'Informatique de Grenoble
Université Grenoble Alpes, CNRS, Grenoble INP
http://convecs.inria.fr

MARS workshop 2024
System-on-Chip & Validation

- **SoC (System-on-Chip) architectures**

- **Priority:** Bug hunting

- **Security:** Resource isolation

- **Modern SoCs:** too complex for traditional validation methodologies (directed tests, constrained random test)
Model-based System-on-Chip Testing

New industrial inclination: Modeling for Testing

“Modeling without testing is meaningless”

Two modeling tasks: behavior & test scenario

PSS (Portable-test and Stimulus Standard)

- Behavior: *actions*
  ordered by *flow objects* (buffer, state, stream)
- Test scenario: *verification intent* (VI)
  composition of actions with process calculi operators
- Focus on VI: behavior only to fill gaps in the VI

Similar to academic *conformance testing*

- Generation of test cases for a *behavior* and *test purpose*
- Supported by CADP (LNT language) and TESTOR
Outline

- Hardware Resource isolation for SoC architectures
- Modeling the behavior in LNT and PSS
- Modeling the test scenarios and generating tests
- Conclusion
Hardware Resource Isolation

- Mechanism to ensure a program or IP cannot access data or functionalities not intended for it
- ARM PSA (Platform Security Architecture)
  - Security: Secure/Non-secure (TrustZone)
  - Privilege: Privileged/Non-privileged (elevation levels EL₀-EL₃)
LNT code for TARGET

1 process TARGET [Read, Grant_Read, Reject_Read, Write, Grant_Write, 
   Reject_Write, Protection, Grant_Protection, 
   Reject_Protection: Bus] (id: ip) is

   require not (source (id));

   var d, e: data, s, t, u: security, p, q, r: privilege, o, other: ip in
     d := data1;  — default value
     s := non_secure; p := non_privileged;  — lowest protection level
   loop
     select
       Read (?o, id, ?t, ?q) where source (o);
       if valid_access (s, t, p, q) then
         Grant_Read (o, id, d)
       else
         Reject_Read (o, id)
       end if
     ...  — communication between other IPs on the shared interconnect
     [] Read (?other, ?o, ?any security, ?any privilege) 
       where (o != id) and source (other)
     ...
   end select
   end loop
end var
end process

1 process LNT per IP

Rendezvous on the same gates (actions)
LNT Behavior Modeling Results

- Several **equivalent** models (hiding source IDs)
  - 8 sources (stable configuration) and 1 target
    182 states, 558 transitions, and 99 labels
  - 1 source (changing configuration) and 1 target
    52 states, 268 transitions, and 39 labels

- Model checking of temporal logic properties
  (e.g., each request is followed by a response, illegal requests are rejected, ...)

- Large state spaces for more than 1 target
PSS Behavior Modeling

Inspired by the 1 source/1 target LNT model

- 21 actions
- 2 state FOs (source and target)
- 9 stream FOs (to emulate rendezvous)
- Constraints to indicate unchanged state fields

Tedious, error-prone, > 500 lines, huge state space
1.7 billion states, 14 billion transitions, 7000 labels
Monolithic PSS Behavior Modeling

- Monolithic, complex state
- No streams
- 11 actions
- Less modular
- More constraints
- Bisimilar to LNT model (after renaming and hiding)

```haskell
action t_grant_read {
  input  system_state in_state;
  output system_state out_state;

  constraint in_state.initial == false;
  // Move from Read to Idle
  constraint in_state.sstate == read;
  constraint out_state.sstate == idle;
  // Check protection
  constraint (in_state.source_sec == secure) ||
              (in_state.target_sec == non_secure);
  constraint (in_state.source_priv == privileged) ||
              (in_state.target_priv == non_privileged);
  // Maintain source fields
  constraint out_state.source_sec == in_state.source_sec;
  constraint out_state.source_priv == in_state.source_priv;
  constraint out_state.source_data == in_state.source_data;
  // Maintain target fields
  constraint out_state.target_sec == in_state.target_sec;
  constraint out_state.target_priv == in_state.target_priv;
  constraint out_state.target_data == in_state.target_data;
  constraint out_state.new_sec == in_state.new_sec;
  constraint out_state.new_priv == in_state.new_priv;
}
```
Test Generation from Test Scenarios

- Test scenario:
  partial ordering of some actions from the behavior

- Two test scenarios illustrating both methodologies
  (two more test scenarios in the paper)

- Differences of the methodologies:

<table>
<thead>
<tr>
<th></th>
<th>PSS methodology</th>
<th>Conformance testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test scenario</td>
<td>Verification intent</td>
<td>Test purpose</td>
</tr>
<tr>
<td>Test generation</td>
<td>Backward inference</td>
<td>Forward exploration</td>
</tr>
</tbody>
</table>
Test 2: Interleaving of all Responses

1. process PURPOSE_2 [LNT]
   2.   Reject_Read,  
   3.   Reject_Write,  
   4.   Reject_Protection,  
   5.   Grant_Read,  
   6.   Grant_Write,  
   7.   Grant_Protection,  
   8. TESTOR_ACCEPT: none] is
   9. par
      10. Grant_Read
      11. || Grant_Write
      12. || Grant_Protection
      13. || Reject_Read
      14. || Reject_Write
      15. || Reject_Protection
      16. end par;
      17. loop TESTOR_ACCEPT end loop
      18. end process

   action intent_2 {
      19. t_grant_read Grant_Read;  
      20. t_grant_write Grant_Write;  
      21. t_grant_protection Grant_Protection;  
      22. t_reject_read Reject_Read;  
      23. t_reject_write Reject_Write;  
      24. t_reject_protection Reject_Protection;  
      activity {  
      25. schedule{
            26. Grant_Read;  
            27. Grant_Write;  
            28. Grant_Protection;  
            29. Reject_Read;  
            30. Reject_Write;  
            31. Reject_Protection;  
      32. }  
      33. }
   34. }

PSS: only shortest tests without repetitions

LNT: all tests with coverage guarantees
Test 4: Access data with different protection

```plaintext
1  process PURPOSE_4 [Read, Grant_Read, Write, Grant_Protection: Bus,
2                    TESTOR_ACCEPT, TESTOR_REFUSE: none] is
3    var s,t: security, p,q: privilege, d: data in
4       Grant_Protection (?any ip, ip0, ?s, ?p)
5       Write (?any ip, ip0, s, p, ?d);  —— same s and p as in the previous line
6       select
7           —— refuse any further rendezvous on gate Grant_Protection
8       Grant_Protection (?any ip, ip0, ?s, ?p); loop TESTOR_REFUSE end loop
9       []  —— accept all other rendezvous
10      null
11     end select;
12    Read (?any ip, ip0, ?t, ?q) where (s != t) or (p != q);
13    Grant_Read (?any ip, ip0, d);  —— access data with different security and privilege levels
14    loop TESTOR_ACCEPT end loop
15   end var
16 end process
```

Cumbersome and error-prone to express in PSS
Conclusion

This talk: Compare modeling & testing approaches of PSS and LNT

Formal modeling in the hardware design domain

- Modeling is considered the future for test generation
- Building complete system models is not envisaged

PSS enables modeling in view of test generation but does *not* enable conformance testing

**Perspective**: Combine both worlds
- formal model-based conformance testing as front-end
- PSS test execution as back-end
Thank You

For Further Information

PSS
https://accellera.org/downloads/standards/portable-stimulus

CADP
https://cadp.inria.fr

CONV/ECS
https://convecs.inria.fr