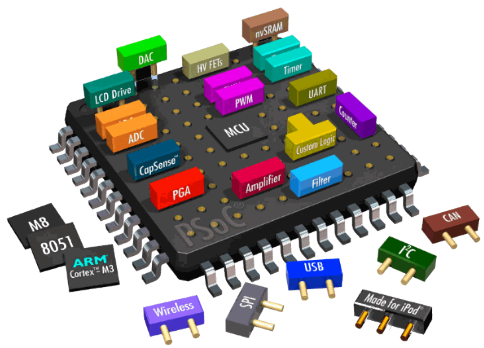
Testing Resource Isolation for SoC Architectures

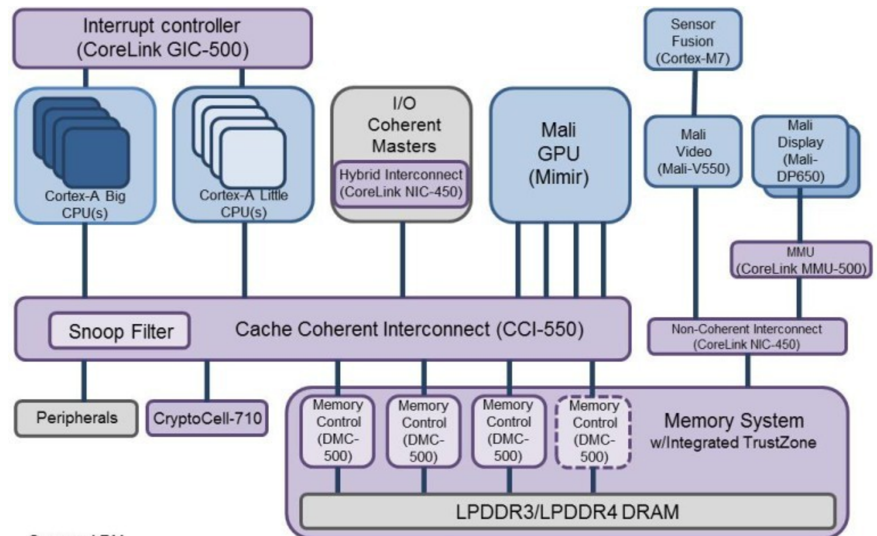
Philippe Ledent   Radu Mateescu
Wendelin Serwe
INRIA – Laboratoire d'Informatique de Grenoble
Université Grenoble Alpes, CNRS, Grenoble INP
http://convecs.inria.fr

MARS workshop 2024

System-on-Chip & Validation

- SoC (System-on-Chip) architectures





Source: ARM

- 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

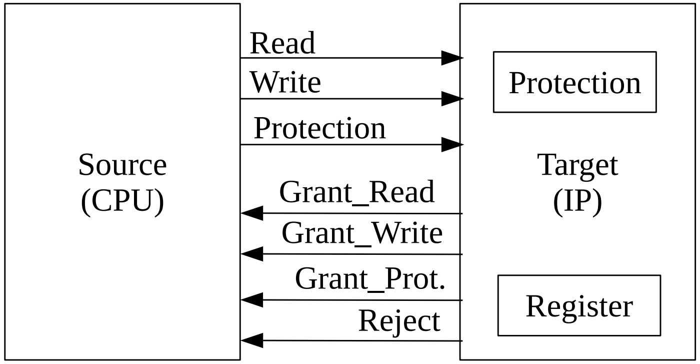
- New industrial inclination: Modeling for Testing
  "Modeling without testing is meaningless"
- **Two** modeling tasks: **behavior** & **test scenario**
- **PSS** (Portable-test and Stimulus Standard)
  - Behavior: *actions*
    ordered by *flow objects* (buffer, state, stream)
  - Test scenario: *verification 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

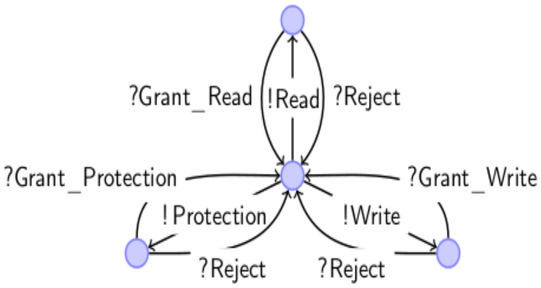
Outline

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

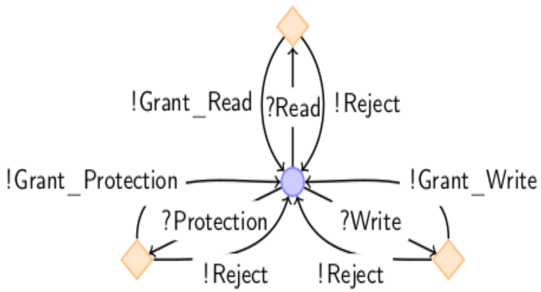
Hardware Resource Isolation

- 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 EL_0-EL_3)





source



target

LNT code for TARGET

```
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

```
action t_request_read {
  input  target_state in_state;
  input  request_read_stream in_stream;
  output target_state out_state;

  constraint in_state.initial == false;
  // Idle -> Read
  constraint in_state.sstate  == idle;
  constraint out_state.sstate == read;
  // save stream data
  constraint out_state.tx_sec        == in_stream.sec;
  constraint out_state.tx_priv       == in_stream.priv;
  // Maintain fields
  constraint out_state.data      == in_state.data;
  constraint out_state.sec       == in_state.sec;
  constraint out_state.priv      == in_state.priv;
  constraint out_state.tx_data   == in_state.tx_data;
  constraint out_state.next_sec  == in_state.next_sec;
  constraint out_state.next_priv == in_state.next_priv;
}
```

- 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)

```
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:

	PSS methodology	Conformance testing
Test scenario	Verification intent	Test purpose
Test generation	Backward inference	Forward exploration

Test 2: Interleaving of all Responses

LNT

```
process PURPOSE_2 [
              Reject_Read,
              Reject_Write,
              Reject_Protection,
              Grant_Read,
              Grant_Write,
              Grant_Protection,
              TESTOR_ACCEPT: none] is
   par
         Grant_Read
      || Grant_Write
      || Grant_Protection
      || Reject_Read
      || Reject_Write
      || Reject_Protection
   end par;
   loop TESTOR_ACCEPT end loop
end process
```

PSS

```
action intent_2 {
  t_grant_read          Grant_Read;
  t_grant_write         Grant_Write;
  t_grant_protection    Grant_Protection;
  t_reject_read         Reject_Read;
  t_reject_write        Reject_Write;
  t_reject_protection   Reject_Protection;
  activity {
    schedule{
          Grant_Read;
          Grant_Write;
          Grant_Protection;
          Reject_Read;
          Reject_Write;
          Reject_Protection;
    }
  }
}
```

- PSS: *only shortest* tests *without repetitions*
- LNT: *all* tests with *coverage guarantees*

Test 4: Access data with different protection

```
process PURPOSE_4 [Read, Grant_Read, Write, Grant_Protection: Bus,
                   TESTOR_ACCEPT, TESTOR_REFUSE: none] is
   var s,t: security, p,q: privilege, d: data in
      Grant_Protection (?any ip, ip0, ?s, ?p)
      Write (?any ip, ip0, s, p, ?d); -- same s and p as in the previous line
      select
         -- refuse any further rendezvous on gate Grant_Protection
         Grant_Protection (?any ip, ip0, ?s, ?p); loop TESTOR_REFUSE end loop
      [] -- accept all other rendezvous
         null
      end select;
      Read (?any ip, ip0, ?t, ?q) where (s != t) or (p != q);
      Grant_Read (?any ip, ip0, d); -- access data with different security and privilege levels
      loop TESTOR_ACCEPT end loop
   end var
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

CONVECS

https://convecs.inria.fr