Assertion-Based Functional Consistency Checking Between TLM and RTL Models

Mingsong Chen
Shanghai Key Lab of Trustworthy Computing
East China Normal University

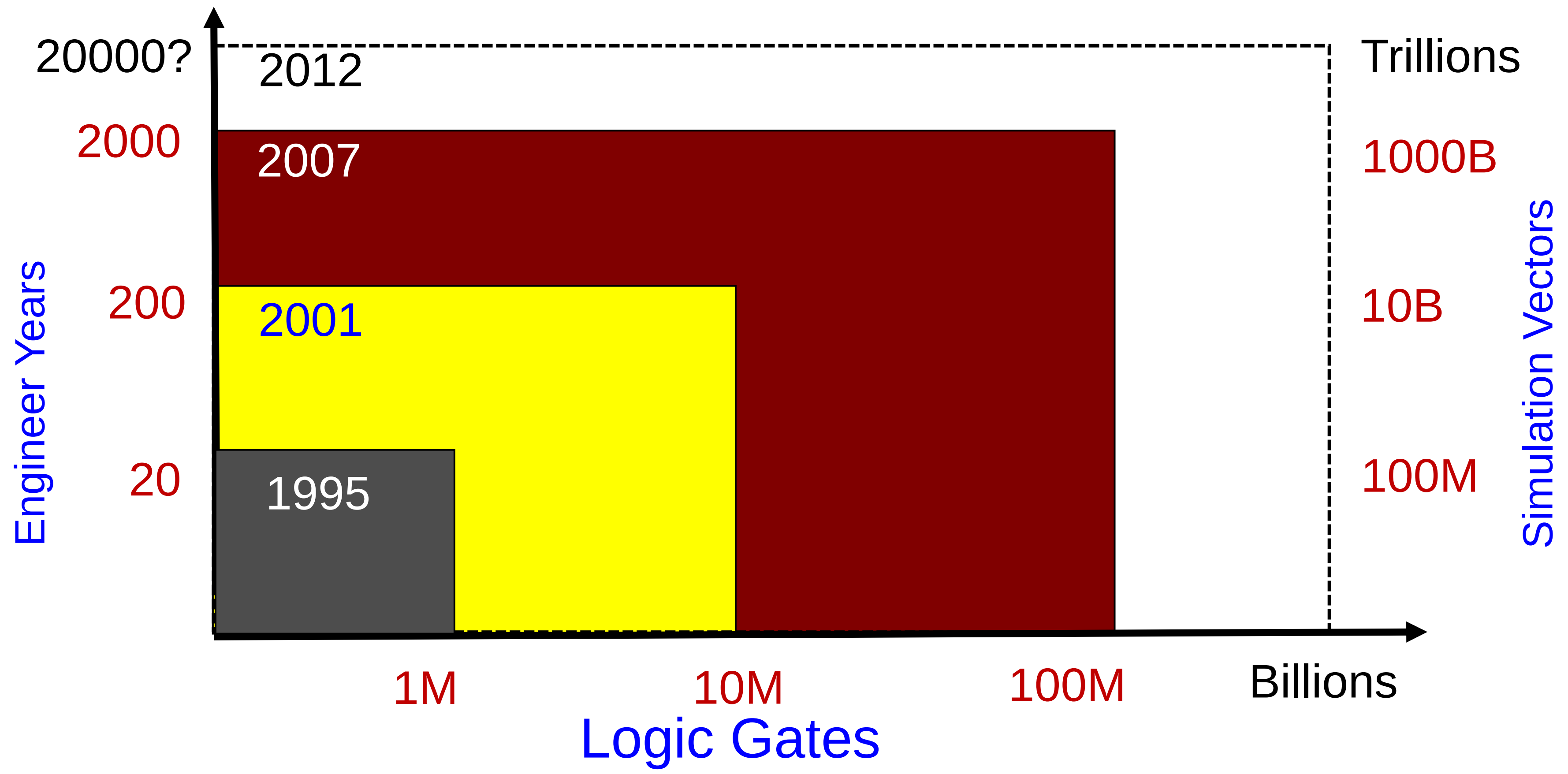
Prabhat Mishra
Computer and Information Science and Engineering
University of Florida, USA

This work was partially supported by NSF CAREER award 0746261, NSF of China 61202103, Open Project of SCERC 2012002, and SHTDP of China 2011AA010101.

Outline

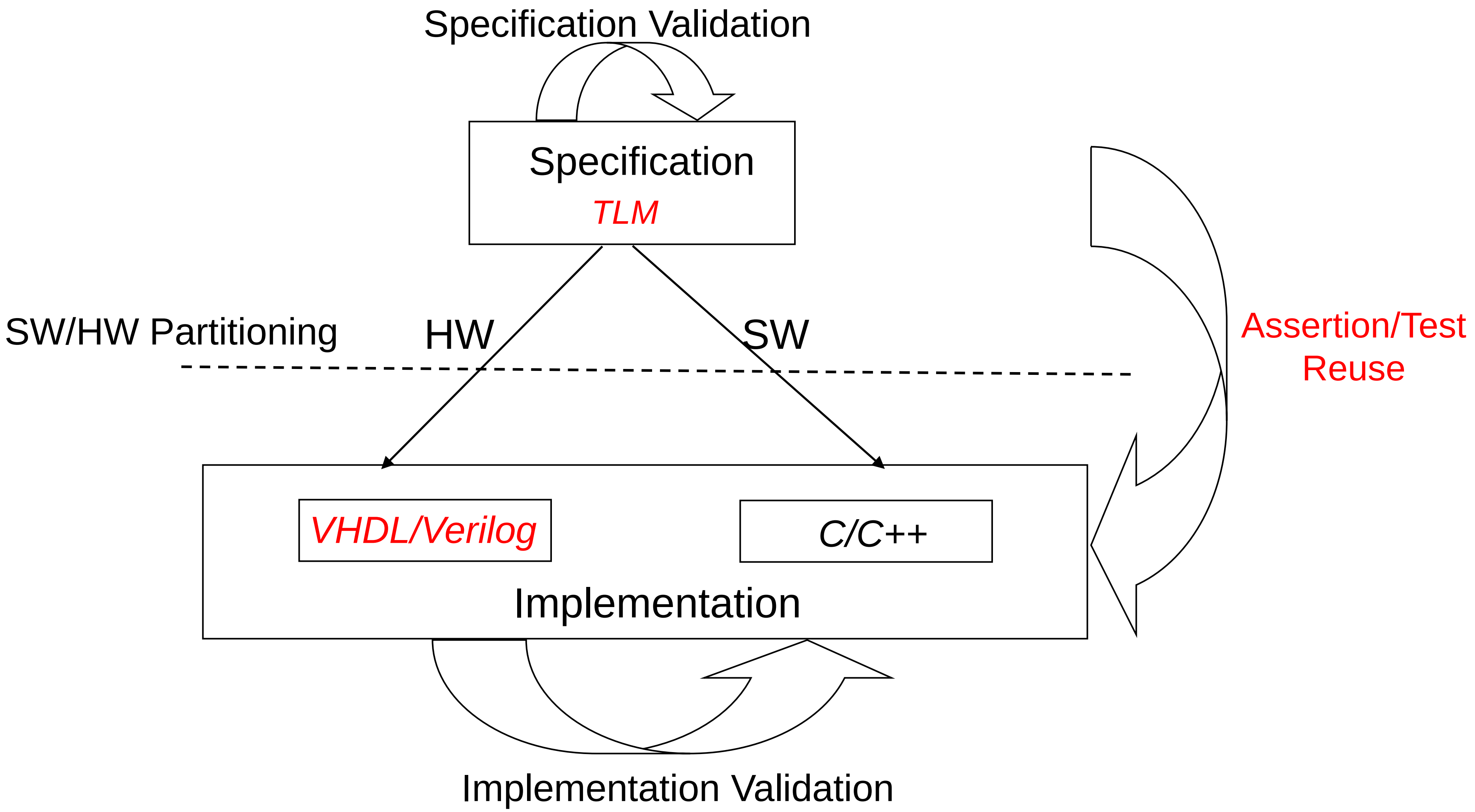
- Introduction
- Related Work
- Assertion-Based Consistency Checking
  - Automatic TLM Assertion Generation
  - Refinement of TLM Assertions/Tests
  - Assertion-Based Functional Consistency Checking
- Experiments
- Conclusions

Functional Validation of SoC Designs



SoC Validation is becoming a major bottleneck.
Up to 70% time and resources are used.

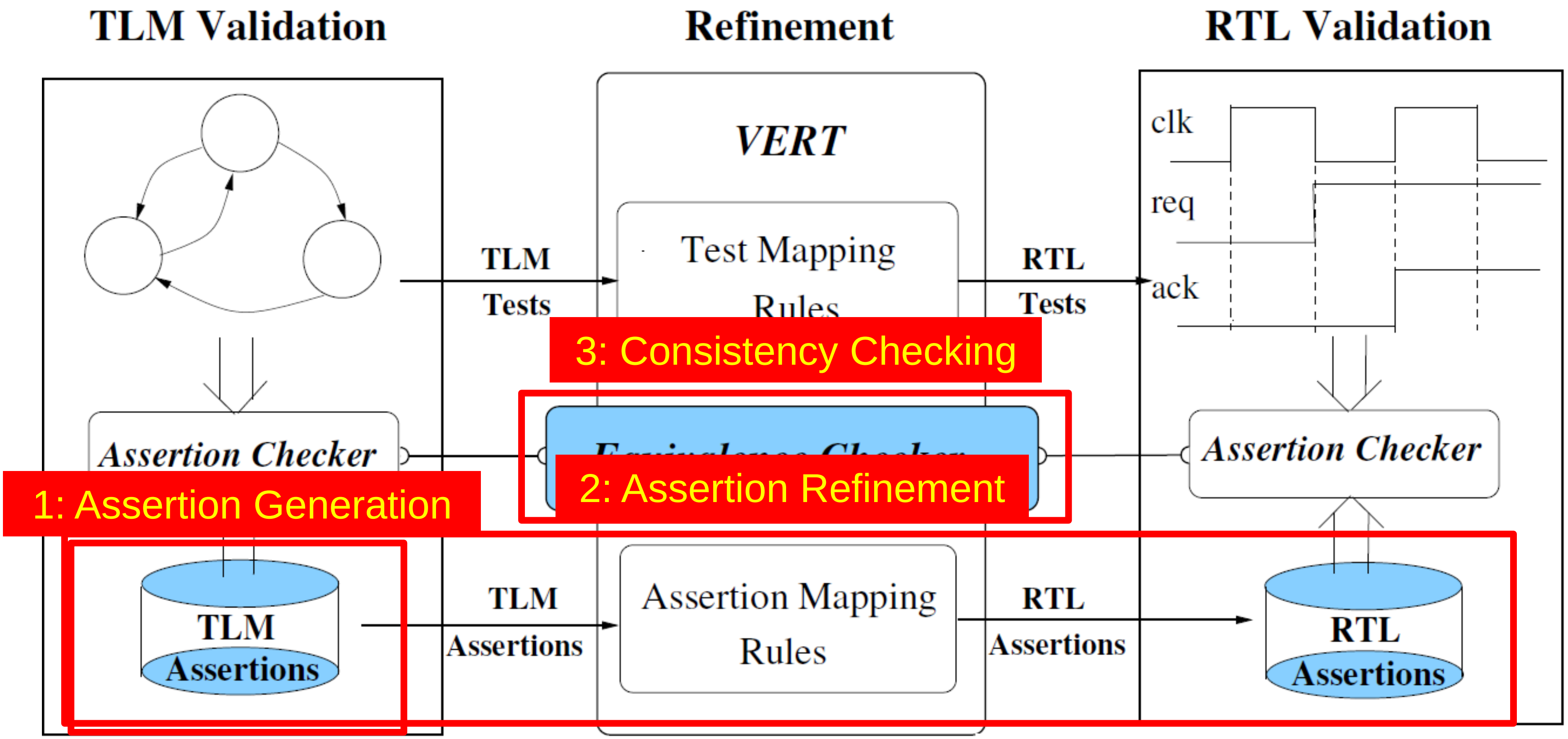
SoC Design and Validation Flow



Related Work

- Transactor-based dynamic verification methods
    - TLM tests can be used in TLM-RTL co-simulation
    - Based on event order without timing information
    - Assertions applied on TLM designs only
- PSL-based Verification approaches
    - Increase the design observability
    - Take advantages of formal techniques
- Few of them investigate the relations of TLM and RTL assertions

Overview of Our Framework



Basic idea: If a TLM test can exercise some TLM assertions, then its RTL counterpart can also activate the corresponding RTL assertions.

Three Issues

- How to define the set of TLM assertions for observing functional scenarios?
  - TLM fault models for automatic assertion generation
- How to reuse TLM validation effort?
  - TLM assertion/test refinement
- How to use the correlation between TLM and RTL assertions for consistency checking
  - Assertion-based consistency checking criteria

Automatic TLM Assertion Generation

- Since we focus on the activation of functional scenarios, we use the following PSL statement pairs to detect whether the sequence P will happen finally.

```
Prop1:   assert   eventually!    p;
Prop2:   cover    (p);
```

  - Prop1 asserts that the sequence *p* must “eventually hold strongly" during the simulation.
  - Prop2 is used to record the assertion coverage during the simulation by using verification directive “cover”.

Automatic TLM Assertion Generation

- We define a set of fault models. Each fault indicates a required “design behavior” which may be violated during the system design.
  - **Transaction data fault model** deals with the possible value assignment for each part of a transaction data.

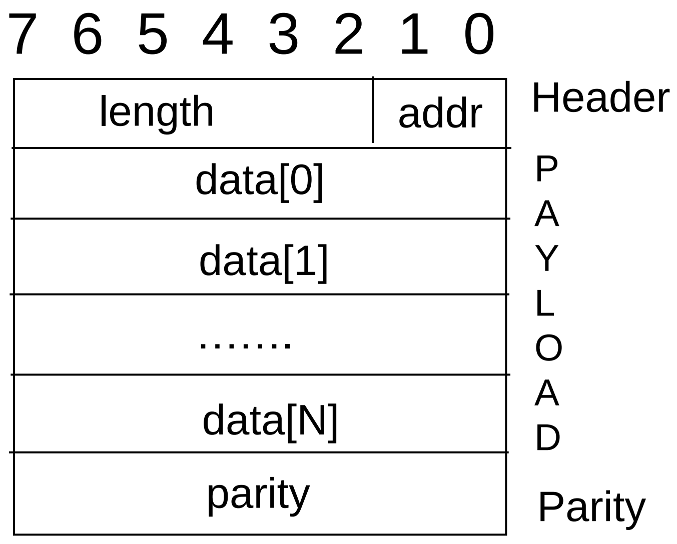
```
// The second bit of “packet.parity” can be 1.
 assert eventually! (packet.parity==2);
 cover (packet.parity==2);
```

  - **Transaction flow fault model** handles the controls (e.g., if-then-else) along a transaction flow.

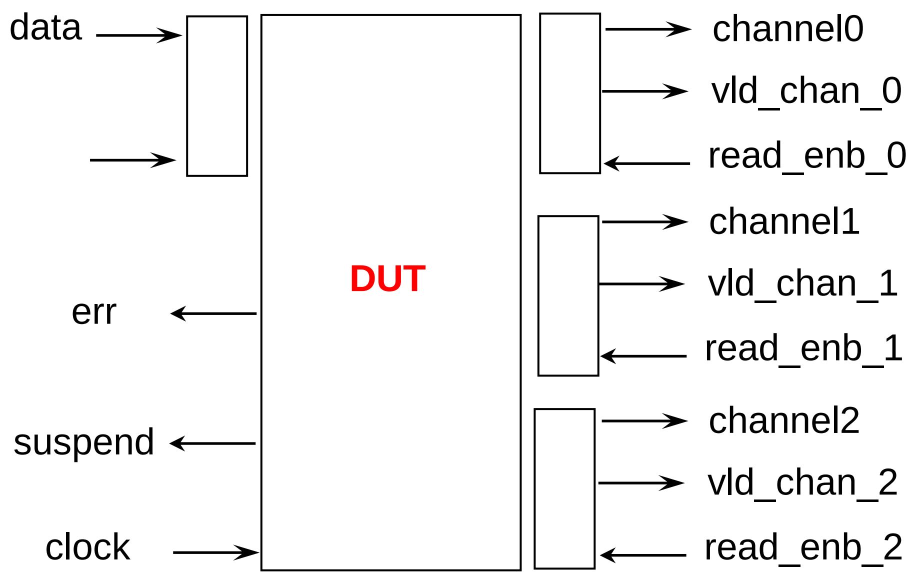
```
// The condition packet.to_chan=1 can be true.
 assert eventually! (packet.to_chan==1);
 cover (packet.to_chan==1);
```

Refinement of TLM Assertions/Tests

- TLM design is significantly different from its RTL implementation in port definition, internal structure and timing details.



Packet Structure



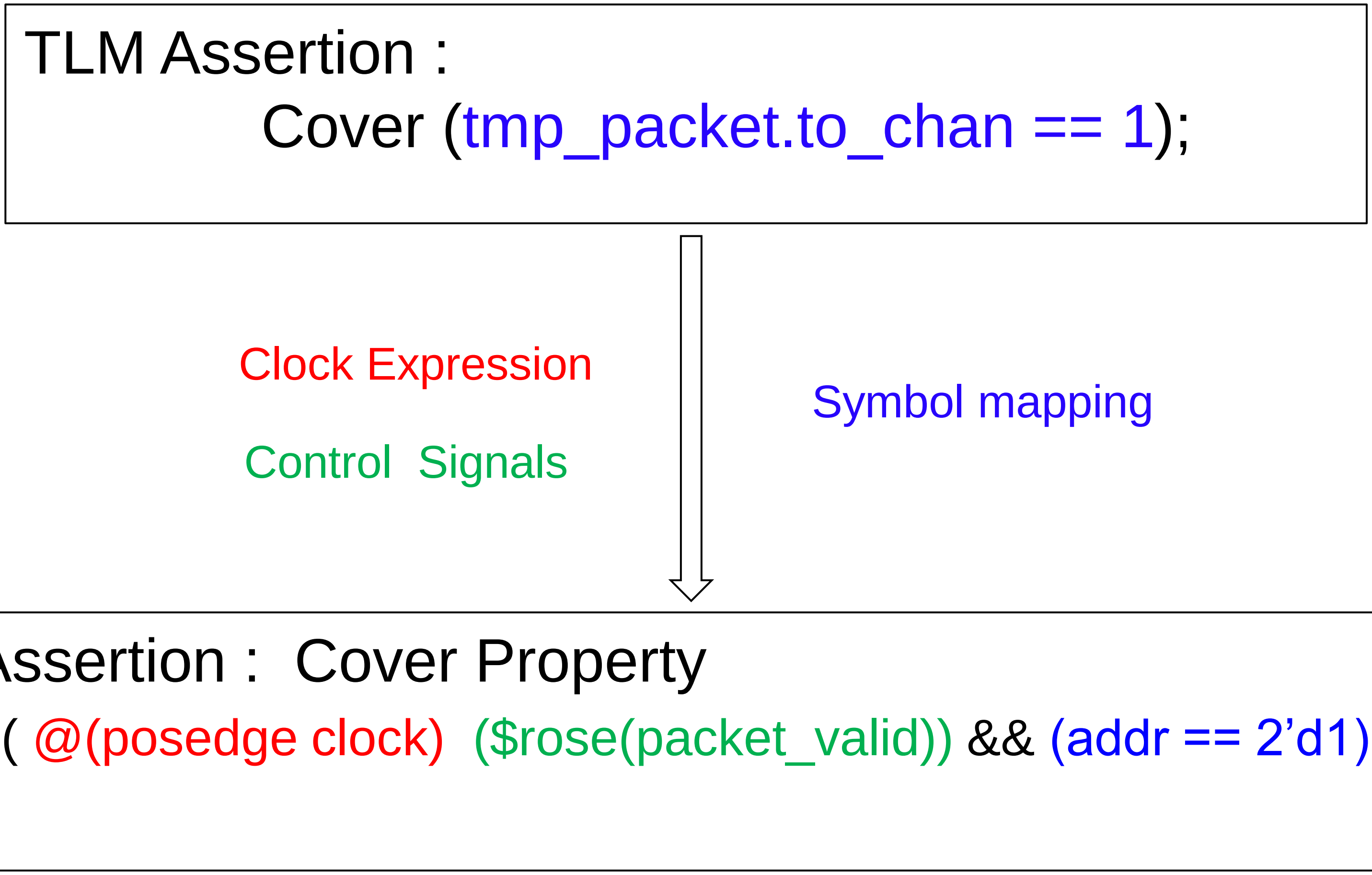
Refinement of TLM Assertions/Tests

- We developed the **Assertion Refinement Specification (ARS)** which contains the rules to guide the assertion refinement. Generally an ARS contains two parts:
  - **Symbol Mapping** specifies the name and type mapping between TLM variables and RTL signals
  - **Assertion Refinement Rules** specify control signals and timing information for RTL assertions.

```
SYMBOL_MAPPING
    bit[1:0] addr = tmp_packet.to_chan;
    ……
END_SYMBOL_MAPPING

ASSERTION_SPEC
    `set_clock (posedge clock);
     ……
    `control   tmp_packet.to_chan
              @  $rose(packet_valid);
     ……
END_ASSERTION_SPEC
```

Refinement of TLM Assertions/Tests



TLM Assertion :

```
        Cover (tmp_packet.to_chan == 1);
```

RTL Assertion : Cover Property

```
        ( @(posedge clock) ($rose(packet_valid)) && (addr == 2'd1) );
```

Refinement of TLM Assertions/Tests

```
-- Port definition
input.data    (10)     [7:0]   data;
-- TLM and RTL name binding
bit[7:0]   head = {packet_data.payload_size[7:2],
        packet_data.to_chan[1:0]};
--Timing relation
@head   packet_valid = 0'b1;
-- Test translation
head => data;
```

TLM Test

```
p->to_chan=1;
p->payload_sz=2;
p->payload[0]=1;
p->payload[1]=2;

p->parity=10;
```

Data Composition

RTL Test

```
read_enb_0 = 0;
read_enb_1 = 0;
read_enb_2 = 0;                 Initialization
packet_valid = 0;

        reset = 0;
#5   reset = 1;                 Reset Sequence
#20 reset = 0;

#5   packet_valid = 1;          Use of half clock
     data = 8'b00001001;        Name Transformation
#10 data = 8'b00000001;
#10 data = 8'b00000010;
#10 packet_valid = 0;
     data= 8'b00001010;

#10  read_enb_1=1;
#40  read_enb_1=0;              Use of four clocks
$finish;
```

Assertion-based Functional Consistency Checking

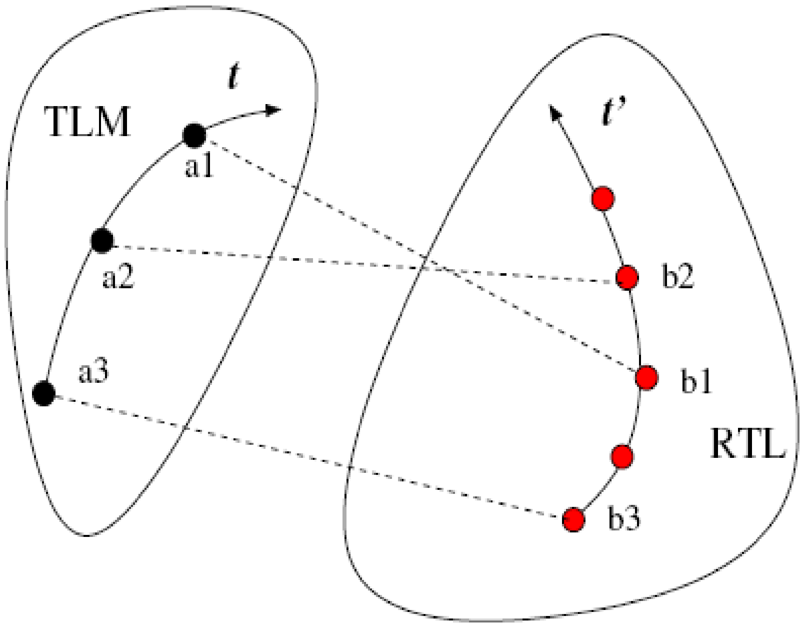
- Since an assertion activation means that a specific functional scenario is covered, the coverage of the assertions indicates the adequacy of the functional validation.
- Given a TLM specification T and its RTL implementation R, by applying TLM tests on T and RTL tests on R, the assertion coverage can be calculated as:

$$T_{coverage} = \frac{\text{\# of exercised TLM assertions}}{\text{Total number of TLM assertions}}$$

$$R_{coverage} = \frac{\text{\# of exercised RTL assertions}}{\text{Total number of RTL assertions}}$$

Assertion-based Functional Consistency Checking

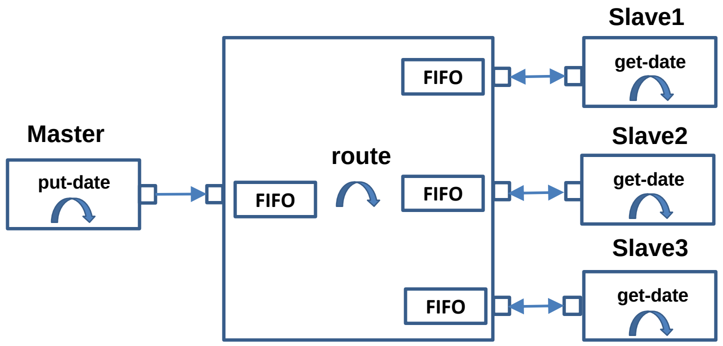
- For a TLM test and its refined RTL version, when applying them on the TLM and RTL designs
  - **Assertion consistent:** For each test, the activated TLM assertions is a subset of the corresponding RTL assertions.
  - **Strongly assertion consistent:** Besides assertion consistency, for each test, it requires that the activation order of assertions is the same.



t and t' are assertion consistent, but they are not strongly assertion consistent.

Case Study 1: A Router Example

- The main function of the router is to parse incoming packets and send them to target slaves.
- By using our tool, 59 TLM assertions are generated.
  - 55 from data fault model
  - 4 from flow fault model
- We select 59 TLM tests from 1000 random TLM tests which can achieve 100% TLM assertion coverage.
- To improve RTL coverage, we derive 2 more directed tests (FIFO overflow + reset).



Case Study 1: A Router Example

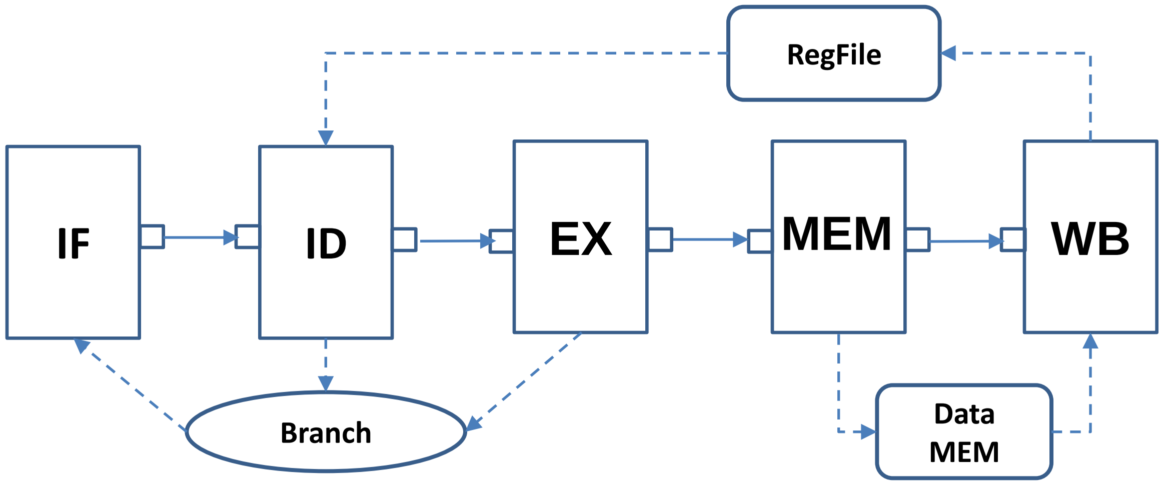
- RTL Coverage using Synopsys VCS Discovery Visualization Environment (DVE) tool

Module	Line	Toggle	FSM	Condition	Path
fifo	76.6%	100%	NA	100%	NA
Port_fsm	95.92%	100%	87.5%	71.88%	100%
router	100%	100%	NA	NA	100%

- The 61 directed RTL tests only need 4 seconds. Running 10000 random tests needs 1057 seconds.
- Found 1 fatal error in the RTL implementation.
  - Try to send the packet to the 4th slave, i.e., to_chan = 3.
- After correcting the error, the TLM and RTL models are strongly assertion equivalent.

Case Study 2: An Alpha AXP Processor

- By using our tool, 163 TLM assertions are generated.
  - 117 from data fault model
  - 46 from flow fault model
- To achieve 100% TLM assertion coverage, 163 TLM tests are selected from 3000 random TLM tests.



Case Study 2: An Alpha AXP Processor

- RTL implementation Coverage of 163 directed tests using Synopsys DVE tool.

Module	Line	Toggle	FSM	Condition	Path
IF_stage	100%	68.82%	NA	100%	100%
ID_stage	100%	80.00%	60.00%	100%	100%
EX_stage	100%	52.94%	NA	100%	100%
MEM_stage	100%	74.19%	NA	100%	100%
WB_stage	100%	78.52%	NA	100%	100%
regfile	100%	71.29%	NA	55.56%	100%

- The 163 directed RTL tests only need 15 seconds. Running 50000 random tests needs 1390 seconds.
- The TLM and RTL models are strongly assertion equivalent.

Conclusion

- Raising the abstraction introduces two challenges
  - Functional inconsistency between abstraction levels
  - Increasing validation efforts
- Our work tries to reuse TLM validation effort to enable RTL validation
  - TLM assertion generation/activation
  - TLM-to-RTL assertion/test refinement
  - TLM-to-RTL functional consistency checking
- Experimental results demonstrate that our approach can improve the design quality and significantly reduce the validation effort.

Questions?

Thank you !