FSM-Based Transaction-Level Functional Coverage

Man-Yun Su, Che-Hua Shih, Juinn-Dar Huang, and Jing-Yang Jou

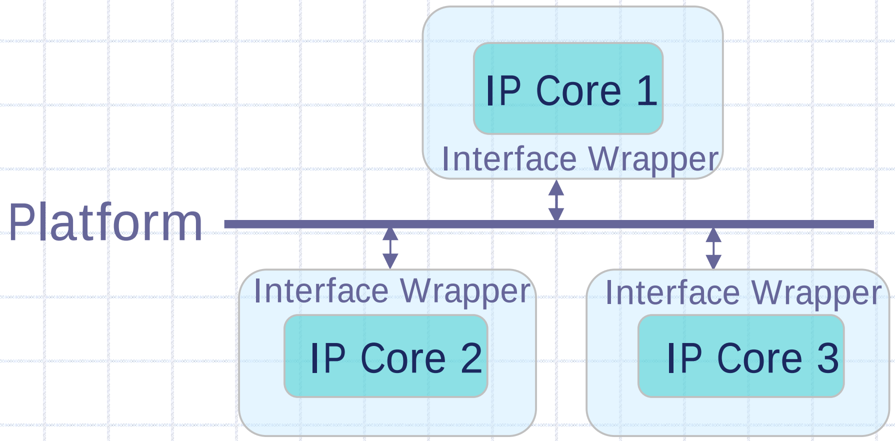
Department of Electronics Engineering,
National Chiao Tung University,
Hsinchu, Taiwan

Outline

- Introduction
- Our Approach
- Experimental Results
- Conclusions

Issues in SoC Design Verification

- Platform-based design methodology with reusable IPs is adopted to accelerate design and verification process
  - platform is based on a specific interface protocol
- Interface compliance verification
  - to guarantee that the interface of each IP conforms to a specific interface protocol



Simulation-Based Functional Verification

- Simulation is the most commonly used method for functional verification
- Coverage metrics: perform a quantitative analysis of simulation completeness
  - measure how well a design has been verified objectively
  - monitor the quality of verification patterns
  - guide direct/random patterns to target the unverified design corners
  - provide a more systematic way to manage the simulation-based verification process

Exploring suitable metrics is an important issue !

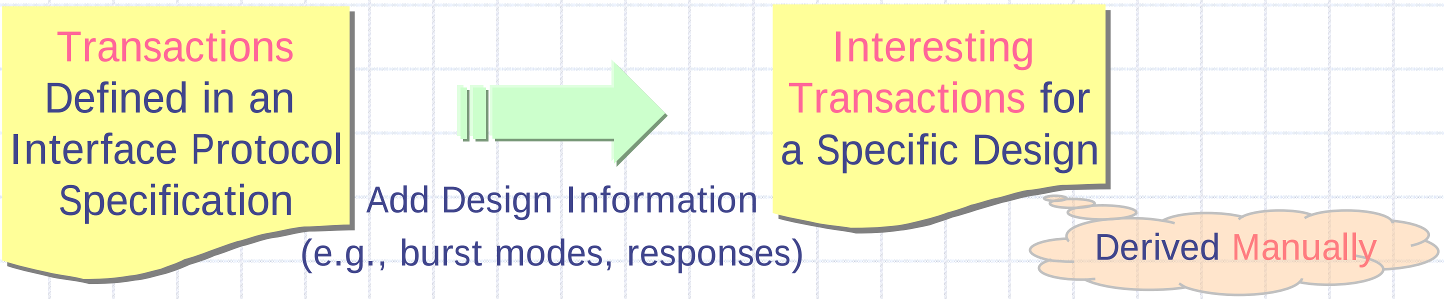
Categories of Coverage Metrics

- Code coverage (or Structural coverage)
  - identify which part of the HDL code has been executed
    - statement coverage, branch coverage, condition coverage, …
  - not sufficient to represent the whole functionality of a design specification
  - not enough for modern complex SoC designs

- Functional coverage
  - focus on the design functionality
  - measure how much of the original design specification has been exercised
  - to further improve verification quality

Functional Coverage for an Interface Design

- Transaction-level functional coverage is one of the commonly used methods

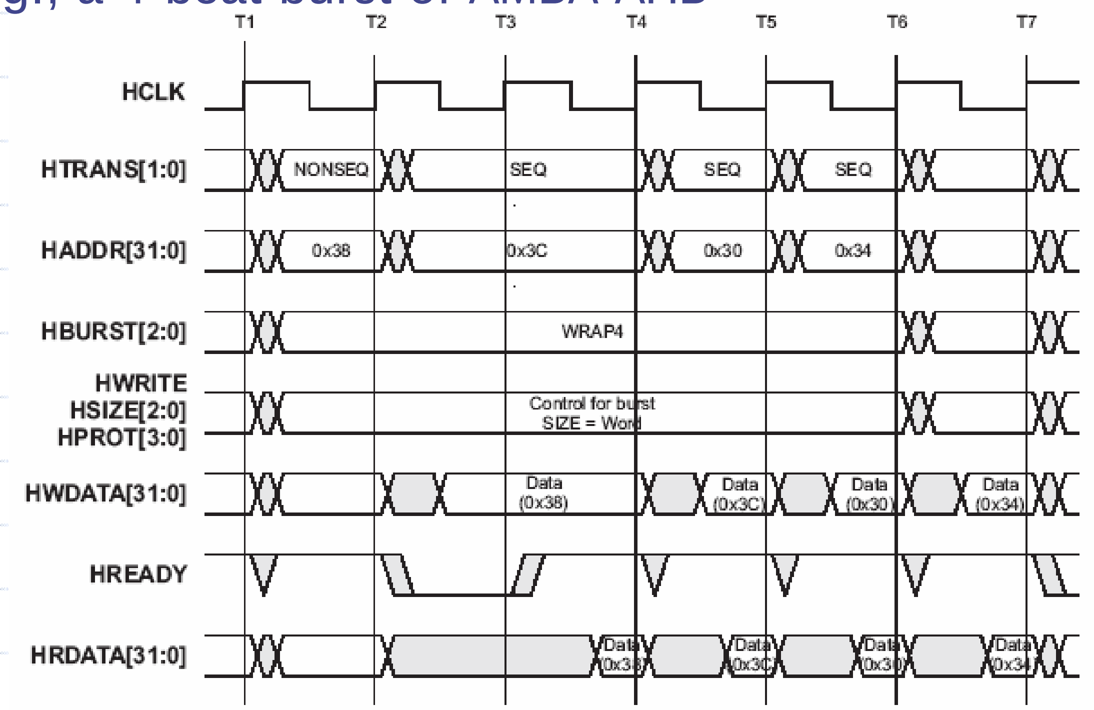


  – a transaction is the transfer of data and control over an interface to perform certain basic operation
    - e.g., 4-beat burst, 8-beat burst, ...

  – interesting transactions for a specific interface design must be derived manually

Motivation

- It is tedious and error-prone for human to specify a transaction if the detailed signal values are required
  - e.g., a 4-beat burst of AMBA AHB



➔ Provide a simple, human-friendly, rigorous, and systematic way to specify transactions at a higher level of abstraction instead of at the signal level

Introduction to Our Approach

- Interface protocol is specified as a spec. FSM
- A transaction can be defined as a specific sequence of state transitions within the spec. FSM
  - raise the level of abstraction to the higher FSM level
  - encapsulate the details of low-level signals
    - e.g., 4-beat burst

      at T1: HTRAN, HBURST, HREADY, HRESP, … =? ➔ S0

      at T2: HTRAN, HBURST, HREADY, HRESP, … =? ➔ S1

      ……
  - put more emphasis on the functionality

Outline

- Introduction
- Our Approach
  - Methodology
  - The Transaction Description Language SOL
- Experimental Results
- Conclusions

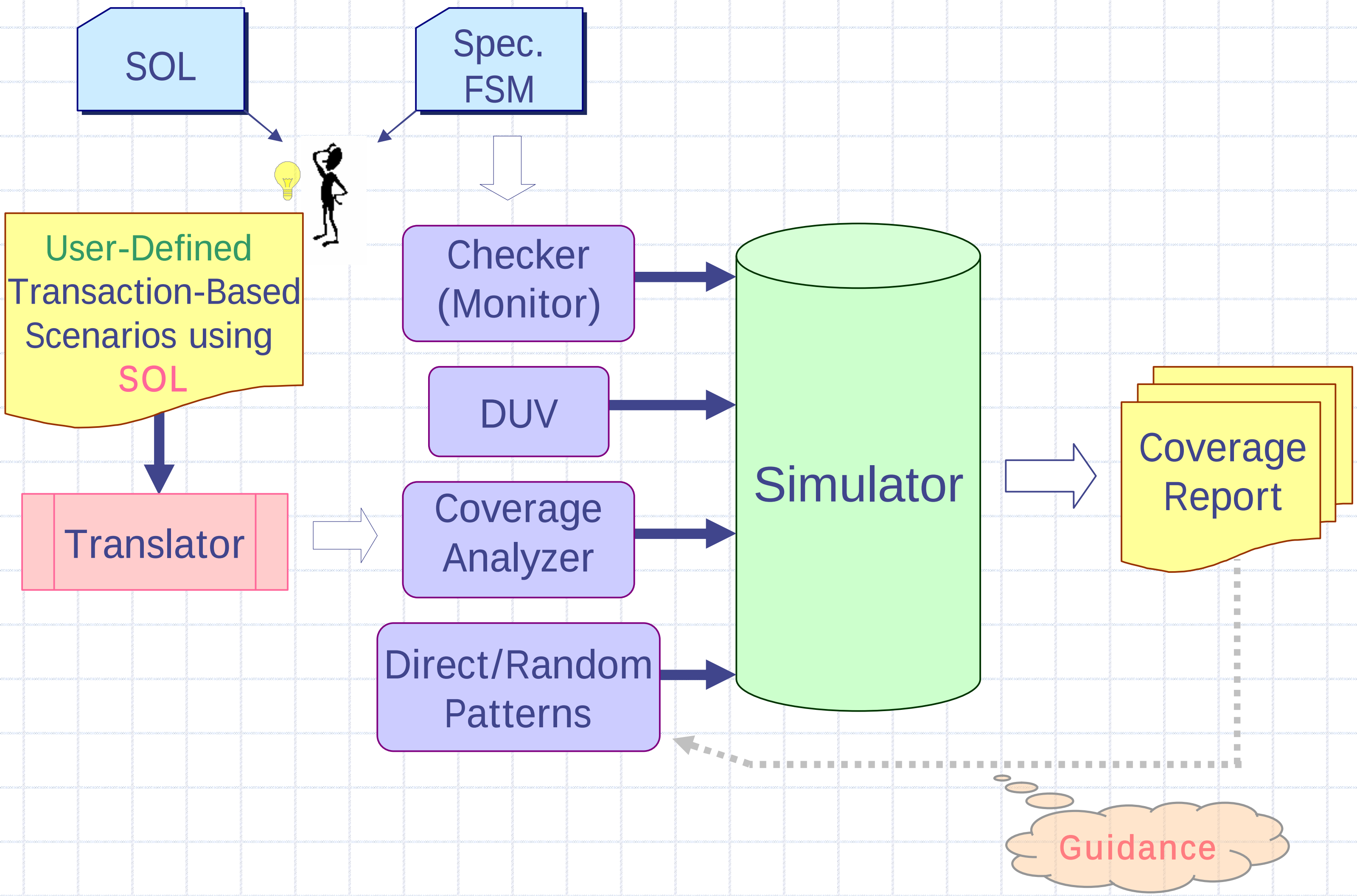
Our Approach

- Develop a transaction description language

  State-Oriented Language (SOL)
  - PSL-like syntax is used to represent sequences of state transitions
  - state is used as the atomic element
  - the expressive power is high for complex transactions

- Proposed verification flow
  - SOL is used to define transactions on spec. FSM manually
  - coverage analyzer is generated automatically
  - transaction-level functional coverage

Flow of Our Verification Methodology



Outline

- Introduction
- Our Approach
  - Methodology
  - The Transaction Description Language SOL
- Experimental Results
- Conclusions

Principles

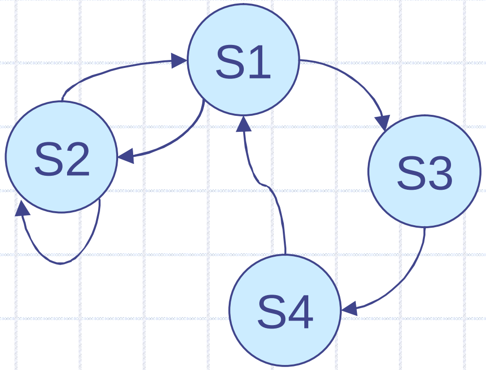
- States are used as basic elements to describe sequences
- Named sequence (=)
  - the left-hand side of the = operator is a synonym for the sequence on the right-hand side
- Sequence names are enclosed in braces { } when referred
- Sequence set is enclosed in angle brackets < > and sequences are separated by commas ,

Expression (1/8)

- Concatenation (;)

FSM

State: S1,S2,S3,S4

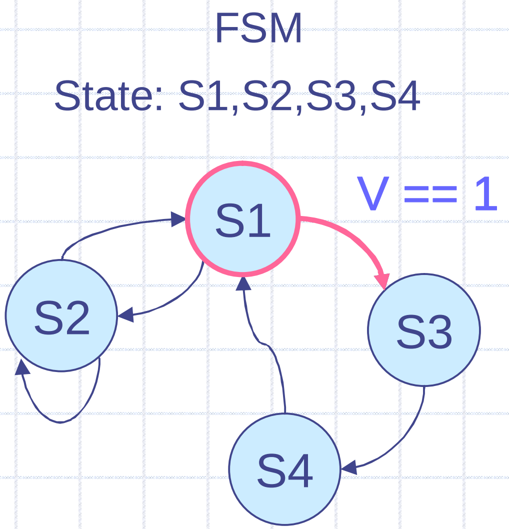


T1 : S1 → S3 → S4 → S1

T1 = { S1 ; S3 ; S4 ; S1 };

Expression (2/8)

- Extra signal qualification (" ")



$\xrightarrow{V == 1}$

T2 : S1 ——→ S3 → S4 → S1

T2 = { S1 "*V* = = 1" ; S3 ; S4 ; S1 };

Expression (3/8)

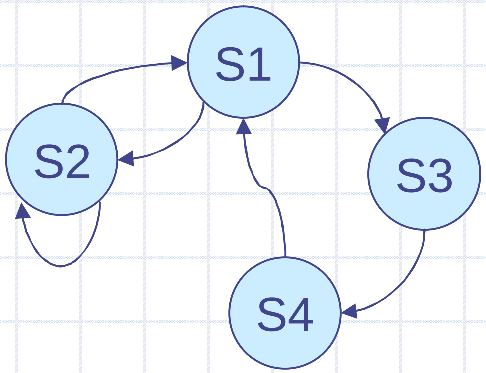
- The repetition operators ([ ]) are used to describe repeated concatenation of the same sequence
- Three types of repetitions
  - consecutive repetition ([*])
  - non-consecutive repetition ([=])
  - goto repetition ([→])

Expression (4/8)

- Consecutive repetition ([*])

FSM

State: S1,S2,S3,S4



T3 : S1 → S2 → S2 → S2 → S1
T3 = { S1 ; S2 ; S2 ; S2 ; S1};
= { S1 ; S2[*3] ; S1 };

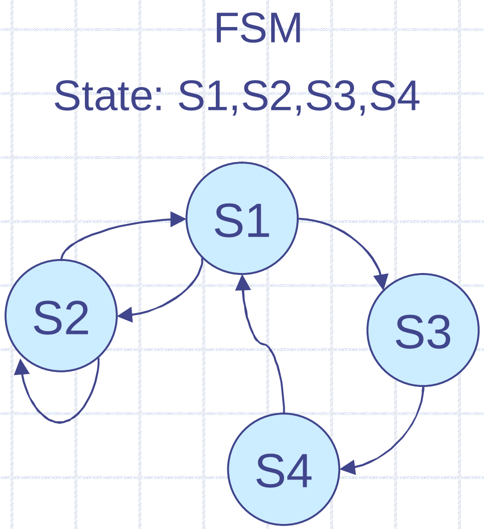
T4 : S1 → S2 (1~5 cycles) → S1
T4 = { S1 ; S2 [*1:5] ; S1 };

T5 : S1 → S2 (ANY # of cycles)
T5 = { S1 ; S2 [*] };

including 0 time

Expression (5/8)

- Sequence AND (&&)



T8 : S1 → . . . (! S3) → S2
→ . . . (! S3) → S2
→ . . . (! S3) → S2

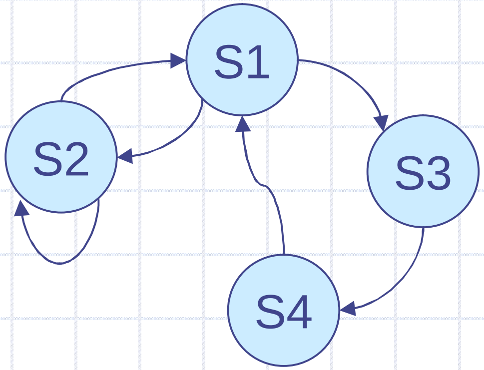
T8 = { S1 ; {S3[=0]}&&{S2[→3]} };

Expression (6/8)

- Sequence OR (|)

FSM

State: S1,S2,S3,S4



T9 : S1→S3→S4→S1 . . . *T1*
OR
S1→S2→S2→S2→S1 . . . *T3*

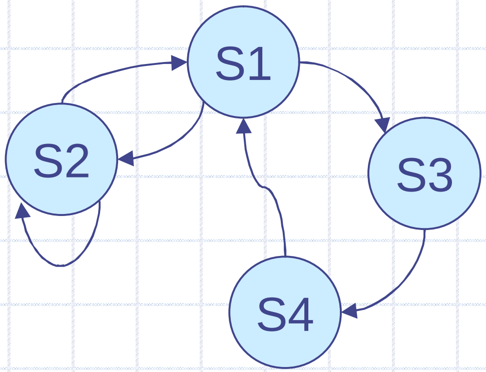
T9 = { {S1;S3;S4;S1}|{S1;S2[*3];S1} };
= { {T1} | {T3} };

Expression (7/8)

- Sequence fusion (:)
  - two sequences overlap each other by one cycle
  - the 2nd sequence starts at the cycle in which the 1st sequence completes

FSM

State: S1,S2,S3,S4



T10 : S1→S3→S4→S1→
S2→S2→S2→S1

➔ S1→S3→S4→S1
S1→S2→S2→S2→S1

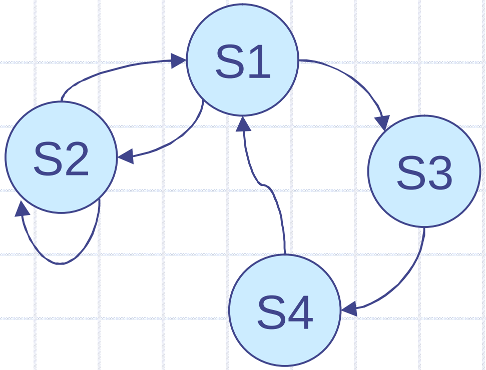
T10 ={ {S1;S3;S4;S1} : {S1;S2[*3];S1} };
={ {T1} : {T3} };

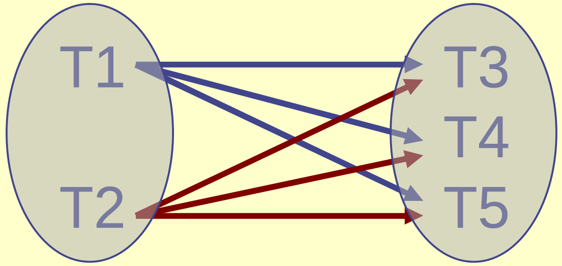
Expression (8/8)

- Sequence set cross (**)

FSM

State: S1,S2,S3,S4





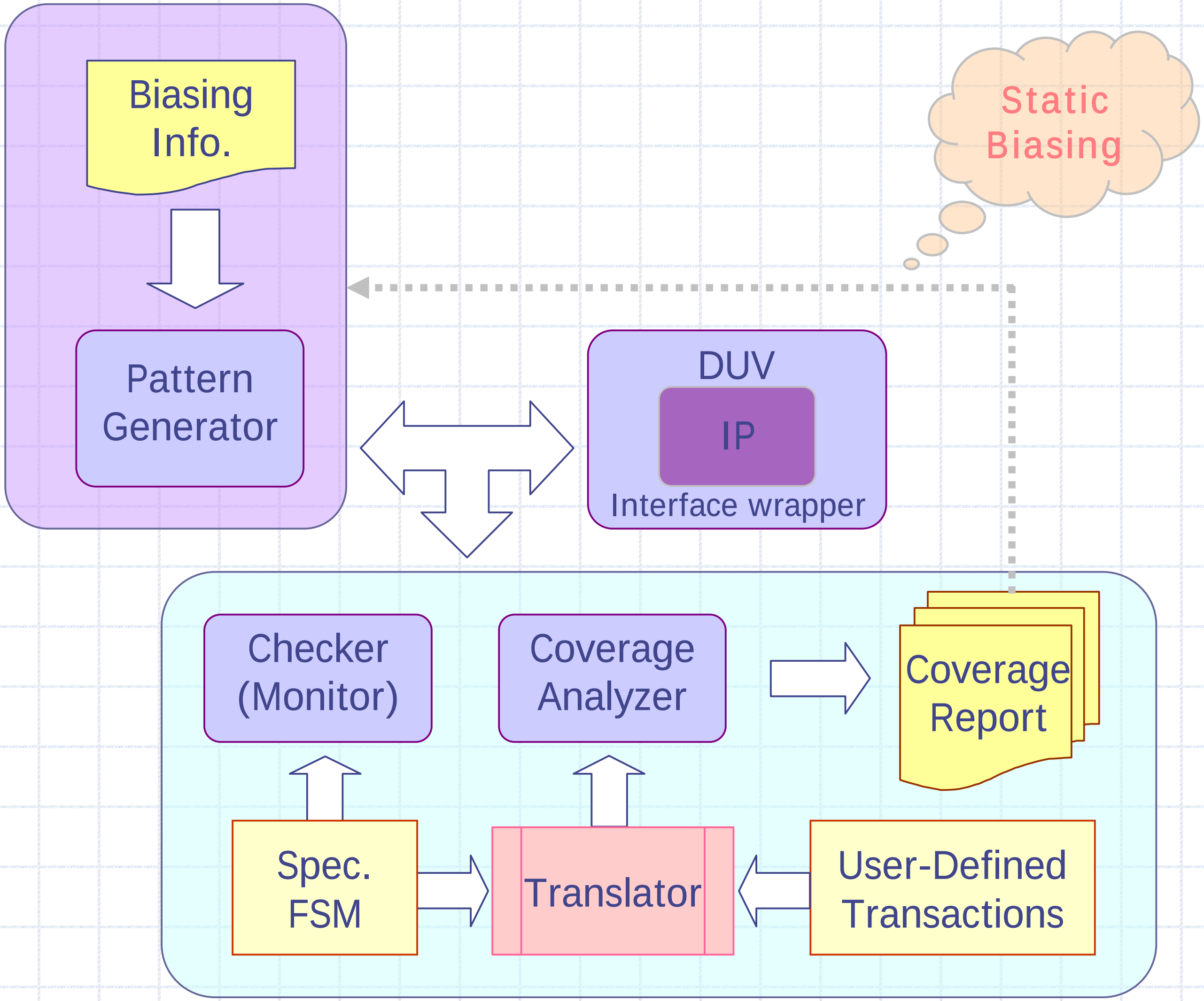
{{T1}:{T3}}; {{T1}:{T4}}; {{T1}:{T5}};
{{T2}:{T3}}; {{T2}:{T4}}; {{T2}:{T5}};

<{T1},{T2}> ** <{T3},{T4},{T5}> ;

Outline

- Introduction
- Our Approach
- Experimental Results
- Conclusions

Experimental Environment



Experimental Result I
Coverage Comparison – Case 1

- Only consider 10 basic read and write transactions
  - e.g., {OneBeatRead}; {OneBeatWrite}; {FourBeatRead}; …

Design	Coverage	# of cycles to reach 100%	Transaction coverage (%)
Convolution	State	12	10 (1/10)
	Transition	47	20 (2/10)
	M-path	102	30 (3/10)
	Transaction	787	100 (10/10)

Experimental Result I
Coverage Comparison – Case 2

- Add 15 more basic transactions with BUSY or WAIT and 25 back-to-back consecutive transactions
  - e.g., {FourBeatWithWAIT}; {FourBeatWithBUSY}; …
    <{Incr},{OneBeat},{FourBeat},{EightBeat},{SixteenBeat}> **
    <{Incr},{OneBeat},{FourBeat},{EightBeat},{SixteenBeat}>;

Design	Coverage	# of cycles to reach 100%	Transaction coverage (%)
Convolution	State	12	4 (2/50)
	Transition	47	8 (4/50)
	M-path	102	12 (6/50)
	Transaction	11135	100 (50/50)

Experimental Result I
Coverage Comparison – Conclusions

- The classical coverage metrics are not capable of providing enough verification quality
- Transaction-level functional coverage
  - put more emphasis on the functionality
  - improve the verification quality

Experimental Result II
Efficiency Improvement

- Increase weights of transitions that may generate BUSY conditions ($bias_1$)
- Adjust weights of 1-beat burst, 4-beat burst, 8-beat burst, and 16-beat burst in a decreasing order ($bias_2$)

Design	Bias	# of cycles to reach 100%	Factor
Convolution	equal weight	11135	1
	$bias_1$	1864	0.167
	$bias_1 + bias_2$	981	0.088

Experimental Result II
Efficiency Improvement – Conclusions

- By exploring coverage reports
  - bias the pattern generator to create more effective patterns to target the unverified corner cases
  - get the same coverage in a shorter time
    - extremely useful for the regression verification
      - the compact and effective patterns are crucial to minimize the required simulation time

Outline

- Introduction
- Our Approach
- Experimental Results
- Conclusions

Conclusions

- A transaction-level functional coverage methodology is proposed for interface compliance verification
- The transaction description language SOL is developed
  - precise and rigorous
  - strong expressive power
  - capable of modeling complex transactions
- A translator is provided to automatically convert the SOL-based transactions into the coverage analyzer
- Experimental results confirm that our methodology can
  - improve the verification quality
  - increase the verification efficiency

Thank You!!

Previous Approaches of Transaction-Level Functional Coverage (1/2)

- M-path coverage
  - model a protocol as a spec. FSM
  - define the M-path as a path which can form a complete bus transfer in the FSM model
    - i.e., a finite sequence of state transitions (a simple transaction)
  - use M-paths as the targets for coverage measurement

- Issues
  - lack expressive power
  - do not consider consecutive transfers

Previous Approaches of Transaction-Level Functional Coverage (2/2)

- CWL-based approach
  - Component Wrapper Language: a regular-expression-based syntax is used to describe signal sequences
  - user can construct transaction scenarios and do transaction-level verification

- Issues
  - individual signals must be considered when describing thorough transactions – *signal-level description*
  - syntactically hard to model complex transactions