Outline

• Introduction
  – Formal specifications
  – PSL history, use, and benefits

• PSL basis
  – Boolean, temporal, verification, and modeling layers
  – Weak and strong operators

• Safety and liveness properties

• Sequential extended regular expressions

• PSL Simple subset

• Case study
Formal specifications

• Formal specifications are
  – Easy to read and write
  – Mathematically precise

• What does this mean: “every request must remain asserted until a grant is received?”
  – Which signals are meant?
  – What does “until” mean?
  – ...

• Properties in natural languages are
  – Imprecise, or
  – long and hard to understand, or
  – both
Property Specification Language

- PSL captures the design behavior spread across time in a concise, unambiguous manner.
- While the bottom most layer of a PSL assertion is still a Boolean expression, a property language adds means to express temporal relationships among those expressions and provides operators to capture complex design behaviors.
History

• Based on Sugar (1994)
  – Initially a nice way to write Computation Tree Logic
  – Extended to regular expressions
  – Changed to Linear Time Logic

• Incorporates 10 years of IBM experience in property-based verification

• A rich language with many ways to express any property

• 2005: IEEE Standard P1850
Use of PSL

• For documentation

• As input to a formal verification tool
  – Model checker
  – Theorem prover

• As input for a dynamic verification tool
  – Assertion-based verification (ABV)
Verification

• Does a given design satisfy a given specification?
  – Design, model, ...
  – Specification, property, ...
  – Dynamic verification, static verification

Model checking

- Design, model, ...
- Specification, property, ...
- Dynamic verification, static verification

Simulation

- true/false
Dynamic ABV benefits

• Dynamic ABV identifies bugs at an earlier stage of design cycle
• Simplifying the diagnosis and detection of bugs by localizing the occurrence of a suspected bug to an assertion monitor
• Since the assertions were included in the design as it was created, they steadily monitor the behavior
  – Modules with embedded assertions become self-checking wherever they are later reused
• Providing internal test points in the design
• Assertions are also used at the design boundaries to provide checks for interface behavior
  – Useful when modules from different designers are integrated
  – Such interface-checking assertions make modules more portable
PSL ANATOMY
PSL is a layered language

- Boolean
- Temporal
- Verification
- Modeling
Boolean layer

- Specify logic expressions without specific timing information
- State something about one (simulation) step
  - Evaluate immediately on the first (simulation) step
  - Use the standard syntax of the underlying language
- Example
  
  // A and B are mutually exclusive
  ( not (A and B) )

  // If A holds then B holds
  ( not A ) or B
• Specify when the Boolean expression must be valid, starting from the first (simulation) step

Example

// A and B are always mutually exclusive
always ( not (A & B) )
Verification layer

• Specify how to use the property
  – assert
    • The tool should attempt to prove the property
  – assume
    • The tool may assume the given property is true
  – cover
    • The tool should measure how often the given property occurs during simulation

• Example
  // A and B must always be mutually exclusive
  assert always ( not (A and B) )
Modeling layer

• Allows extra code fragments from the underlying language to be included with the properties to augment what is possible using PSL alone
  – For example, the modeling layer could be used to calculate the expected value of an outputs

• Example

  // If req is asserted, ack must be asserted in the next cycle
  req = readA_req or readB_req;
  assert always ( req -> next (ack and gnt) )

21/06/2012