SystemVerilog Assertions v3.1

Language Status Update
Panel Session

Stephen Meier – Co-Chair SV-AC
Verification Tools Group - Synopsys

Assertion Design Objectives

- Verilog based assertion language
  - Verilog for boolean and bit vector expressions
  - Designed for ease of use with regular expressions, intuitive syntax
- Unified support of verification tools
  - Features defined for simulation, coverage, formal verification and debug applications
- Template/library features for assertion libraries and re-use

Design Working Group – initial draft, semantics defined

Overview of Assertions in SV v3.1

- Immediate and concurrent assertions
- Enhanced regular sequence expressions
- Flexible variable expressions
- Flexible declaration and instantiation
- Common unified sampling semantics
- Template library

Immediate Assertions

- Immediate evaluation during RTL simulation

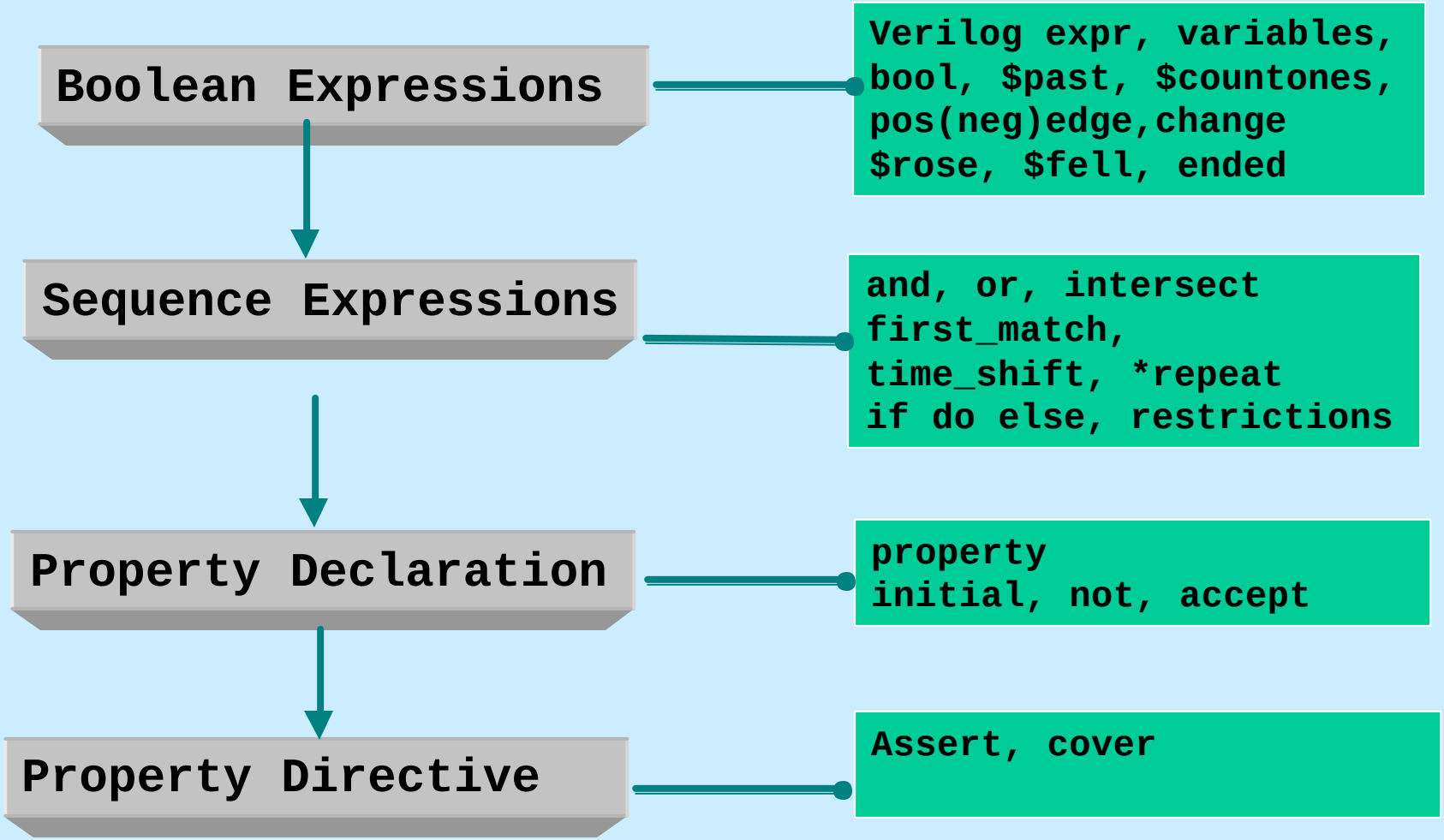
```
assert_mutex : check ( a^b)
                    do $display (“%m passed”) ;
                    else $display (“%m failed”);
```

- Severity level options
- Restricted to only combinational assertion checking
- assert_strobe construct from v3.0 is deprecated

Concurrent Assertions

- Concurrent assertions
  - Sampled at specified Boolean clock expression
  - Consistent cycle semantics across simulation, testbench and formal
- Types of concurrent assertion instantiation
  - RTL module level
    - Declarative at top level
  - RTL procedural
    - Enabling conditions inferred from context
  - External to RTL
    - Directive to bind to module or instance

Hierarchy of Concurrent Assertions



Boolean Expressions - Operators

- All verilog expressions
  - identical 4-state evaluation
- Past values of variables
  - $past (var_name, no_of_cycles)
- Count of ones in a variable
  - $countones (var_name)
- Event detectors
  - posedge, negedge, change (for clocks)
  - $rose, $fell, $stable : detects change of boolean expression
  - ended : detects end of sequence expression

Expression Definitions

- Define a boolean expression using bool, sequence using seq, property using property
  - All types are named and can have parameters
  - Boolean expression not attached to a clock
  - Sequence and property attached to a clock

```
bool mem_req = ( memr[2:1] && meme[1:0] );
event clkev = posedge clk ;
seq @clkev mem_fetch =
        mem_req && rd_mem ; [0:inf]  mem_found;
property cache = @clkev if (cache_hit) mem_fetch ;
assert cache;
```

Sequence Expressions

- Or, And, Intersect

  (req ; [1] read) or (req ; [1] write)

- If Then Else: conditional matching

  if ($fell (frame)) rdy ; [1] data

- Repeat: repeat a sequence multiple times

  seq byte_8 = (byte_req ; [1] byte_read) * [8];

- Signal occurs within a sequence

  occurs data_valid within (req ; [1:8] ack ; [1:8] grant)

Sequence Expressions

- Restriction: restrict length and values

  length [10] within (req ; [1:8] ack ; [1:8] grant)

  istrue stable_sig within (req ; [1:8] ack ; [1:8] grant)

- First Match

  req ;[1] (first_match (ack ;[1 : inf] done))

Variables in Sequences

Basic Variables

- System Verilog variables used in assertion expressions are sampled by sampling clock
- Single variable per assertion across all attempts

Dynamic Variables with let construct

- Used to store data on a per evaluation basis
- Sampled at event in sequence where it is declared
- Ex: Variable x is assigned value of pipe_in at beginning of pipe stage

```
seq e =  if (valid_in)
    let (int x = pipe_in) within [5] (pipe_out1 == (x + 1));
```

Property Declaration

- Define assertion property with name and optional parameters

  property rule1 = @(posedge clk) if (a) (b ; [1]c ;[1] d) ;

- Reset conditions can force success (accept)

  property rule2 = (accept = reset)

  @(posedge clk) if (a) (b ; [1]c ; [1] d);

- Property can be defined to never occur (not)
- Property can be checked for only the first sampling clock tick (initial)

Property Directives

- Assertions instantiated using property directives

  rule1_ins: **assert** rule1;

  input_prop: **assert** @clk (f ; [1]g);

- Directives to guide verification

  assert: Check assertion is always true

  cover : Track coverage results during dynamic simulation

Binding to Design Objects

- Assertions can be declared outside of design and bound to the design
  - Binding to a specified instance

    bind_instance instance1: rule1;
  - Binding to a specified module

    bind_module module_x : rule2;

Assertion Template Support

- Template feature
  - Template definition with parameters
  - Formal arguments can be any of: variables, Boolean and sequence expressions
  - Supports ordered or named list of arguments

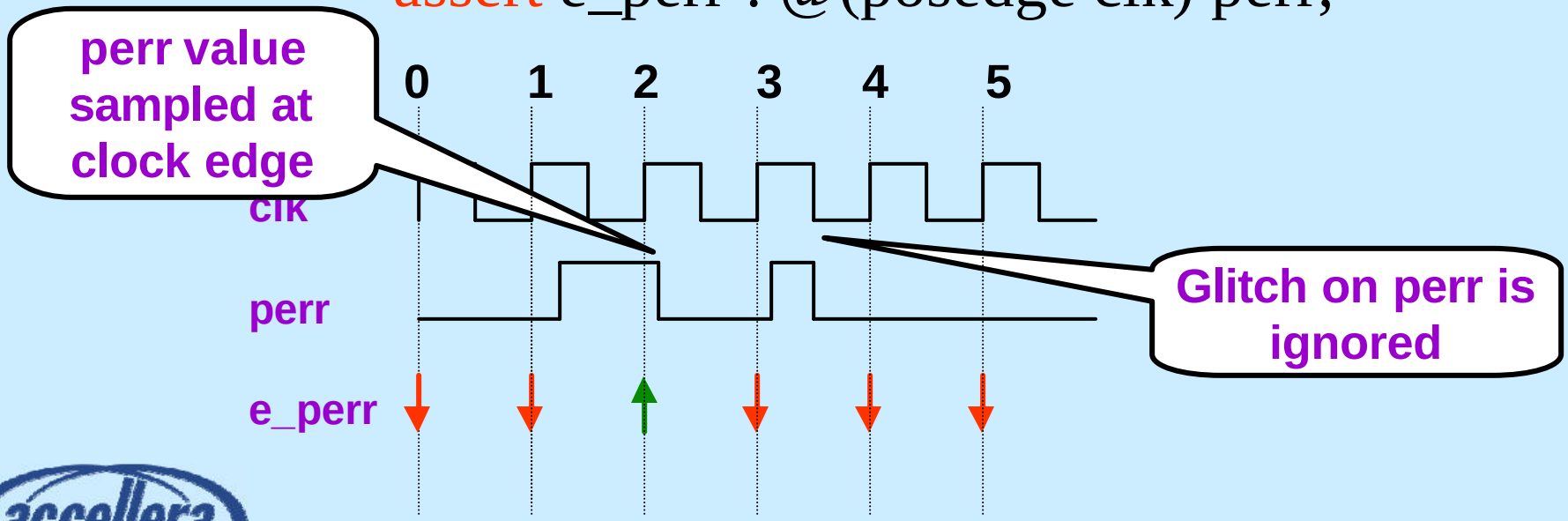
```
template hold ( exp, min = 0, max = 15, clk);
   seq @(clk) e_hold = ( $past(exp) == exp) *
   [min:max] ;
endtemplate

hold hold_instance ( s, 5, , posedge clk1);
```

Concurrent Assertion Sampling

- Sampling defined by value of Boolean expression
- Variables are sampled at the clock edge
  - Variable values are stable values from previous simulation cycle

```
assert e_perr : @(posedge clk) perr;
```



Procedural Semantics

Procedural code example:

```
always @(posedge clk or negedge reset)
  if(reset == 0) do_reset;
  else if (mode == 1)
    case(st)
    REQ: if (!arb)
              if (foo)
                 st <= REQ2;
                   test1: assert ((req && !gnt)*[0:5]; gnt && req; !req);
```

Semantics define automatic translation to declarative form:

```
test1: assert (accept=negedge reset) @(posedge clk)
     if ((mode == 1) && (st == REQ) && (!arb) && (!foo))
        (req && !gnt)*[0:5]; gnt && req; !req);
```

Sampling semantic is same for all concurrent assertions

Overview of Assertions in SV v3.1

- Easy to Use, Expressive language
  - Immediate and concurrent assertions
  - Enhanced regular sequence expressions
  - Per evaluation variables (let)
  - Template declaration and instantiation
- Common Unified Sampling Semantics
  - Cycle based sampling
  - Procedural enabling condition inference
- Embedded in System Verilog
  - In-line assertions in RTL
  - Integration with System Verilog verification features