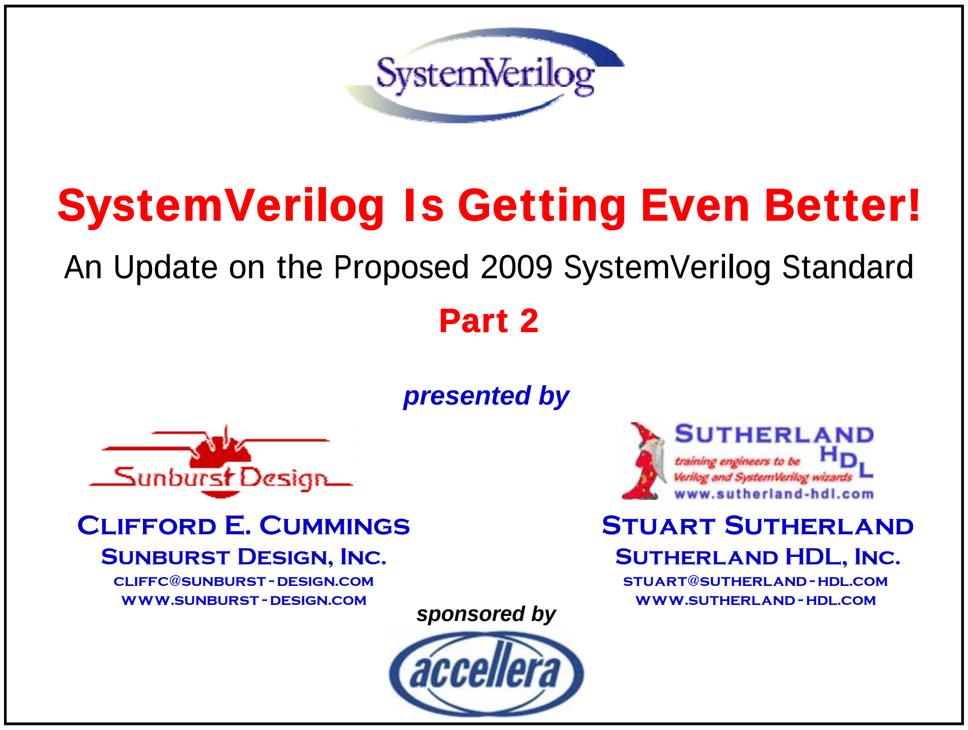
SystemVerilog Is Getting Even Better!

An Update on the Proposed 2009 SystemVerilog Standard, Part II

by Sutherland HDL, Inc., Portland, Oregon



SystemVerilog Is Getting Even Better!

An Update on the Proposed 2009 SystemVerilog Standard

Part 2

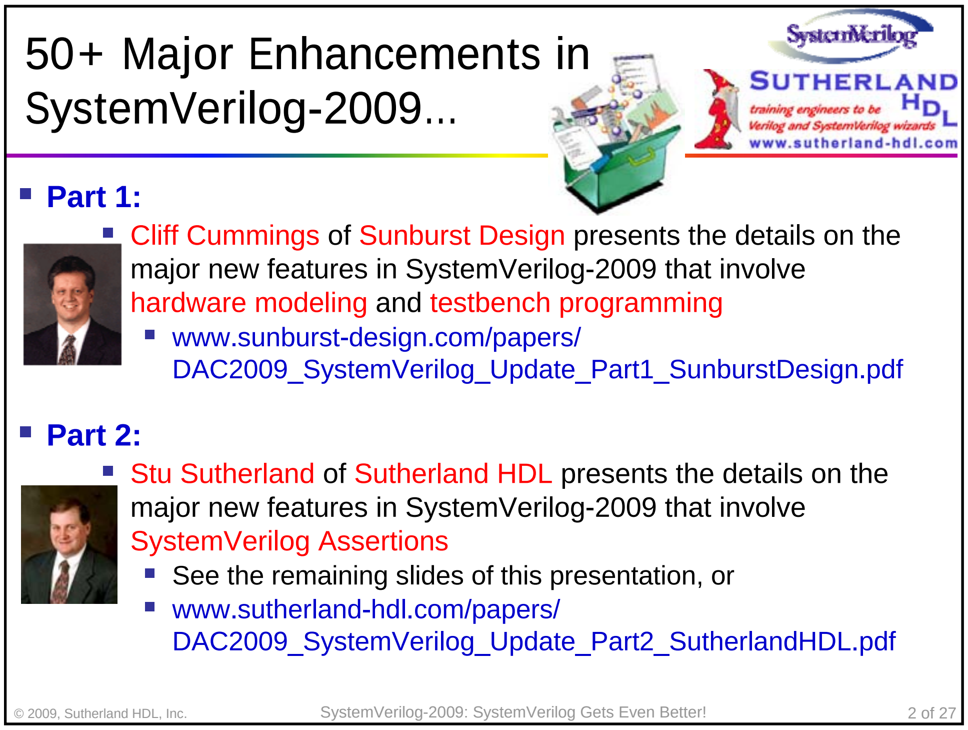
presented by

CLIFFORD E. CUMMINGS
SUNBURST DESIGN, INC.
CLIFFC@SUNBURST-DESIGN.COM
WWW.SUNBURST-DESIGN.COM

STUART SUTHERLAND
SUTHERLAND HDL, INC.
STUART@SUTHERLAND-HDL.COM
WWW.SUTHERLAND-HDL.COM

sponsored by

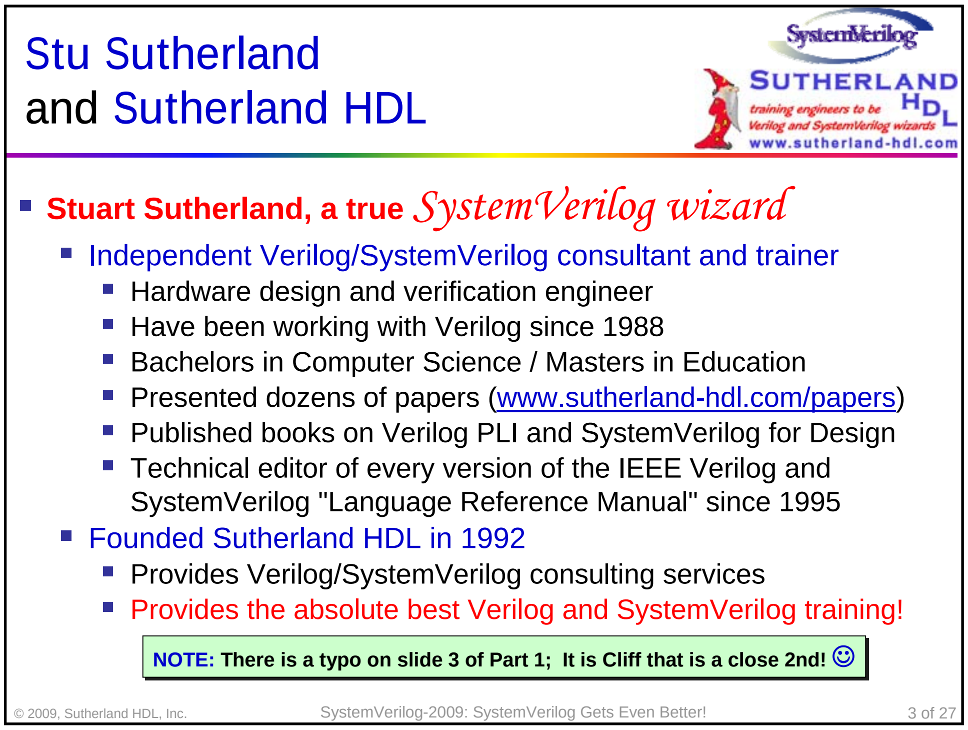
accellera



50+ Major Enhancements in SystemVerilog-2009...

- **Part 1:**
  - Cliff Cummings of Sunburst Design presents the details on the major new features in SystemVerilog-2009 that involve hardware modeling and testbench programming
    - www.sunburst-design.com/papers/DAC2009_SystemVerilog_Update_Part1_SunburstDesign.pdf
- **Part 2:**
  - Stu Sutherland of Sutherland HDL presents the details on the major new features in SystemVerilog-2009 that involve SystemVerilog Assertions
    - See the remaining slides of this presentation, or
    - www.sutherland-hdl.com/papers/DAC2009_SystemVerilog_Update_Part2_SutherlandHDL.pdf

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 2 of 27

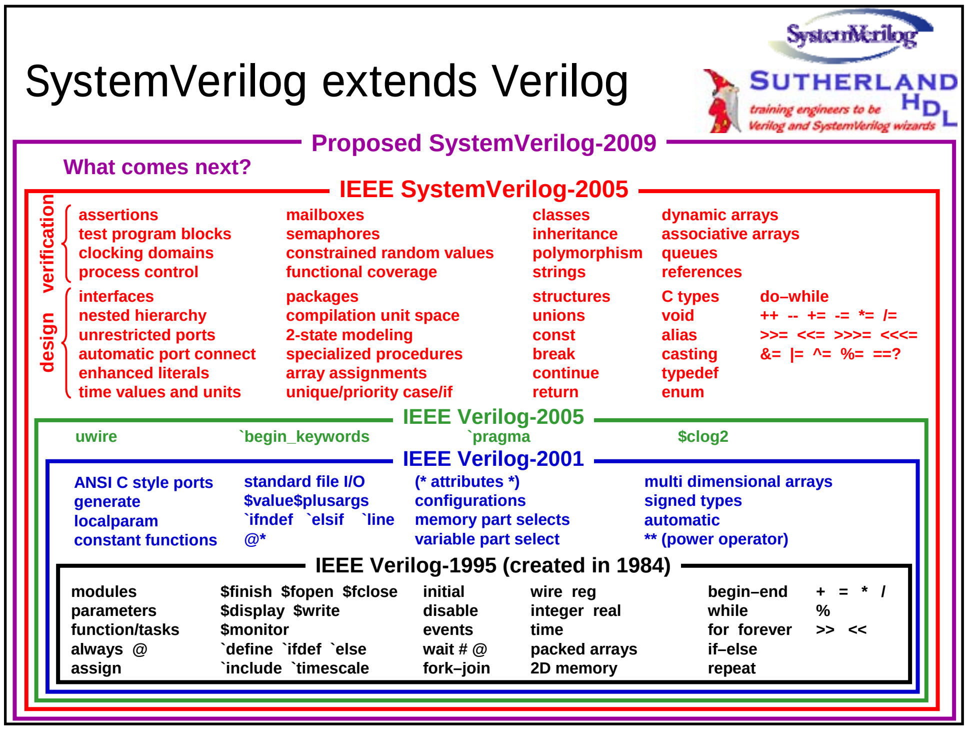


Stu Sutherland and Sutherland HDL

- **Stuart Sutherland, a true *SystemVerilog wizard***
  - Independent Verilog/SystemVerilog consultant and trainer
    - Hardware design and verification engineer
    - Have been working with Verilog since 1988
    - Bachelors in Computer Science / Masters in Education
    - Presented dozens of papers (www.sutherland-hdl.com/papers)
    - Published books on Verilog PLI and SystemVerilog for Design
    - Technical editor of every version of the IEEE Verilog and SystemVerilog "Language Reference Manual" since 1995
  - Founded Sutherland HDL in 1992
    - Provides Verilog/SystemVerilog consulting services
    - Provides the absolute best Verilog and SystemVerilog training!

NOTE: There is a typo on slide 3 of Part 1; It is Cliff that is a close 2nd! ☺

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 3 of 27



SystemVerilog extends Verilog

Proposed SystemVerilog-2009

What comes next?

IEEE SystemVerilog-2005

verification:
assertions, test program blocks, clocking domains, process control
mailboxes, semaphores, constrained random values, functional coverage
classes, inheritance, polymorphism, strings
dynamic arrays, associative arrays, queues, references

design:
interfaces, nested hierarchy, unrestricted ports, automatic port connect, enhanced literals, time values and units
packages, compilation unit space, 2-state modeling, specialized procedures, array assignments, unique/priority case/if
structures, unions, const, break, continue, return
C types, void, alias, casting, typedef, enum
do–while, ++ -- += -= *= /=, >>= <<= >>>= <<<=, &= |= ^= %= ==?

IEEE Verilog-2005

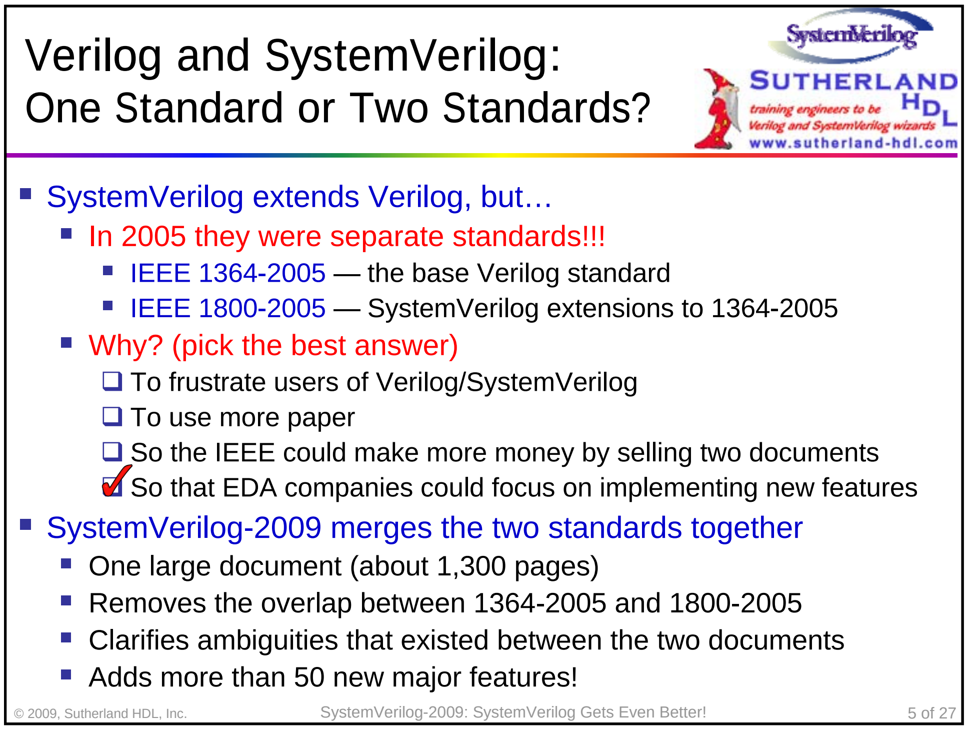
uwire `begin_keywords `pragma $clog2

IEEE Verilog-2001

ANSI C style ports, generate, localparam, constant functions
standard file I/O, $value$plusargs, `ifndef `elsif `line, @*
(* attributes *), configurations, memory part selects, variable part select
multi dimensional arrays, signed types, automatic, ** (power operator)

IEEE Verilog-1995 (created in 1984)

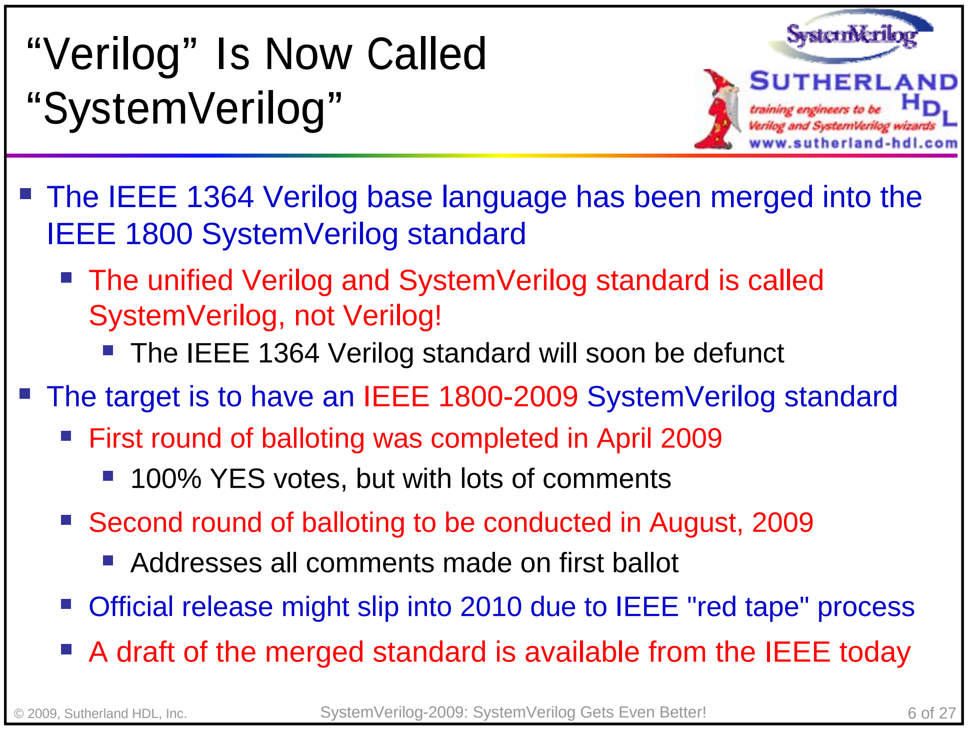
modules, parameters, function/tasks, always @, assign
$finish $fopen $fclose, $display $write, $monitor, `define `ifdef `else, `include `timescale
initial, disable, events, wait # @, fork–join
wire reg, integer real, time, packed arrays, 2D memory
begin–end, while, for forever, if–else, repeat
+ = * /, %, >> <<



Verilog and SystemVerilog: One Standard or Two Standards?

- SystemVerilog extends Verilog, but…
  - In 2005 they were separate standards!!!
    - IEEE 1364-2005 — the base Verilog standard
    - IEEE 1800-2005 — SystemVerilog extensions to 1364-2005
  - Why? (pick the best answer)
    - ☐ To frustrate users of Verilog/SystemVerilog
    - ☐ To use more paper
    - ☐ So the IEEE could make more money by selling two documents
    - [✓ So that EDA companies could focus on implementing new features]
- SystemVerilog-2009 merges the two standards together
  - One large document (about 1,300 pages)
  - Removes the overlap between 1364-2005 and 1800-2005
  - Clarifies ambiguities that existed between the two documents
  - Adds more than 50 new major features!

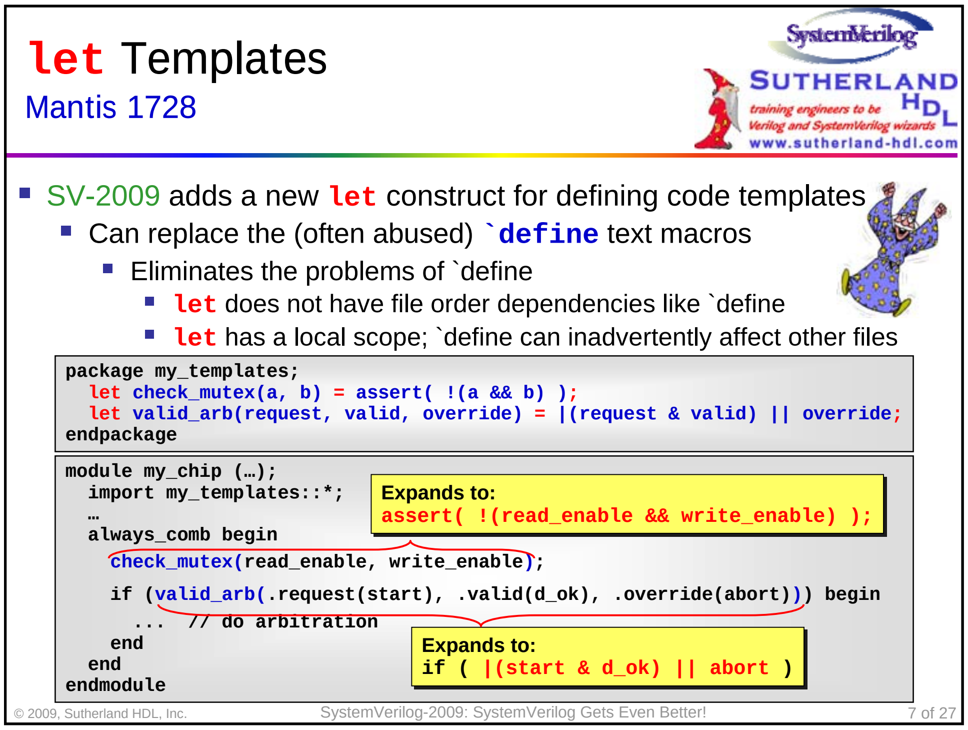
© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 5 of 27



"Verilog" Is Now Called "SystemVerilog"

- The IEEE 1364 Verilog base language has been merged into the IEEE 1800 SystemVerilog standard
  - The unified Verilog and SystemVerilog standard is called SystemVerilog, not Verilog!
    - The IEEE 1364 Verilog standard will soon be defunct
- The target is to have an IEEE 1800-2009 SystemVerilog standard
  - First round of balloting was completed in April 2009
    - 100% YES votes, but with lots of comments
  - Second round of balloting to be conducted in August, 2009
    - Addresses all comments made on first ballot
  - Official release might slip into 2010 due to IEEE "red tape" process
  - A draft of the merged standard is available from the IEEE today

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 6 of 27



let Templates
Mantis 1728

- SV-2009 adds a new `let` construct for defining code templates
  - Can replace the (often abused) `` `define `` text macros
    - Eliminates the problems of `define
      - `let` does not have file order dependencies like `define
      - `let` has a local scope; `define can inadvertently affect other files

```
package my_templates;
  let check_mutex(a, b) = assert( !(a && b) );
  let valid_arb(request, valid, override) = |(request & valid) || override;
endpackage
```

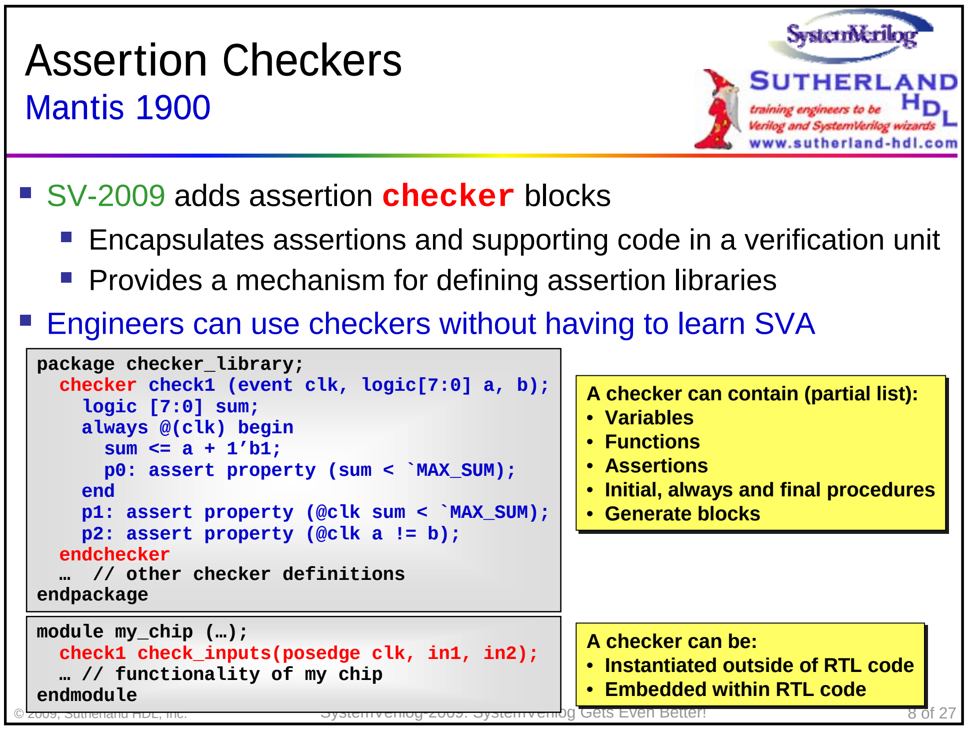
```
module my_chip (…);
  import my_templates::*;
  …
  always_comb begin
    check_mutex(read_enable, write_enable);

    if (valid_arb(.request(start), .valid(d_ok), .override(abort))) begin
      ...  // do arbitration
    end
  end
endmodule
```

Expands to:
`assert( !(read_enable && write_enable) );`

Expands to:
`if ( |(start & d_ok) || abort )`

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 7 of 27



Assertion Checkers
Mantis 1900

- SV-2009 adds assertion `checker` blocks
  - Encapsulates assertions and supporting code in a verification unit
  - Provides a mechanism for defining assertion libraries
- Engineers can use checkers without having to learn SVA

```
package checker_library;
  checker check1 (event clk, logic[7:0] a, b);
    logic [7:0] sum;
    always @(clk) begin
      sum <= a + 1’b1;
      p0: assert property (sum < `MAX_SUM);
    end
    p1: assert property (@clk sum < `MAX_SUM);
    p2: assert property (@clk a != b);
  endchecker
  …  // other checker definitions
endpackage
```

A checker can contain (partial list):
- **Variables**
- **Functions**
- **Assertions**
- **Initial, always and final procedures**
- **Generate blocks**

```
module my_chip (…);
  check1 check_inputs(posedge clk, in1, in2);
  … // functionality of my chip
endmodule
```

A checker can be:
- **Instantiated outside of RTL code**
- **Embedded within RTL code**

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 8 of 27

Assertion Untyped Arguments
Mantis 1601

- SV-2009 adds **`untyped`** as an argument data type
  - Allows explicitly specifying that the formal argument is untyped in sequences, properties, and checkers
  - Allows mixing typed and untyped arguments in any order
    - SV-2005 could have all arguments be untyped, but was ambiguous about having a mix of typed and untyped

```
property (bit clk, logic [63:0] a, untyped b, c);
  ...
endproperty
```

```
logic [63:0] data, address;
sequence s1;
  req ##1 ack;
endsequence

assert property (mclk, data, s1, address);
```

Untyped arguments allow assertions libraries to be more flexible on the actual argument types

NOTE: The new `let` construct can have untyped arguments but the SV-2009 standard does not permit the `untyped` keyword with `let` constructs; This was an oversight in the syntax that will be corrected in a future version of the standard (see Mantis 2835)

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 9 of 27

Assertion Global Clock
Mantis 1681

- SV-2009 adds the ability to specify a global clock definition
  - Simplifies writing assertion definitions for formal verification tools

```
global clocking @(posedge master_clock); endclocking
```

- **There can only be one global clock definition in the entire elaborated model**
- **Can only be declared in a module or interface**

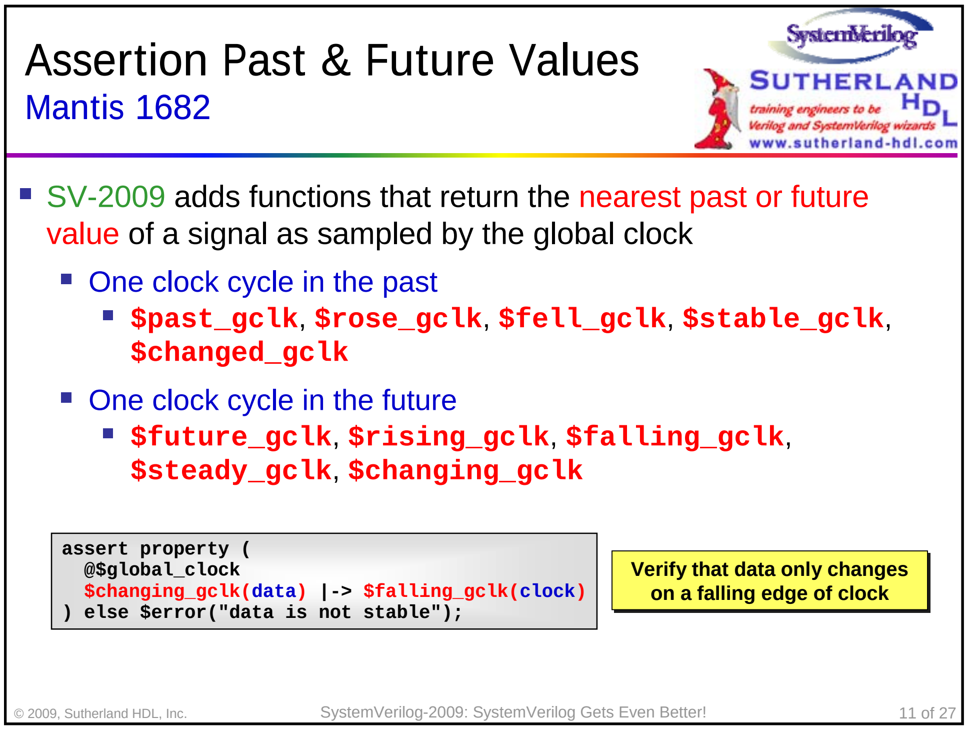
- **`$global_clock`** returns the event expression specified in the global clocking declaration
  - Can be used anywhere that a clocking event can be specified

```
property @($global_clock)
  ...
endproperty
```

```
always @($global_clock) begin
  ...
end
```

- **In simulation, `$global_clock` is the event defined in the global clocking declaration**
- **In formal verification, `$global_clock` is the primary system clock**

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 10 of 27



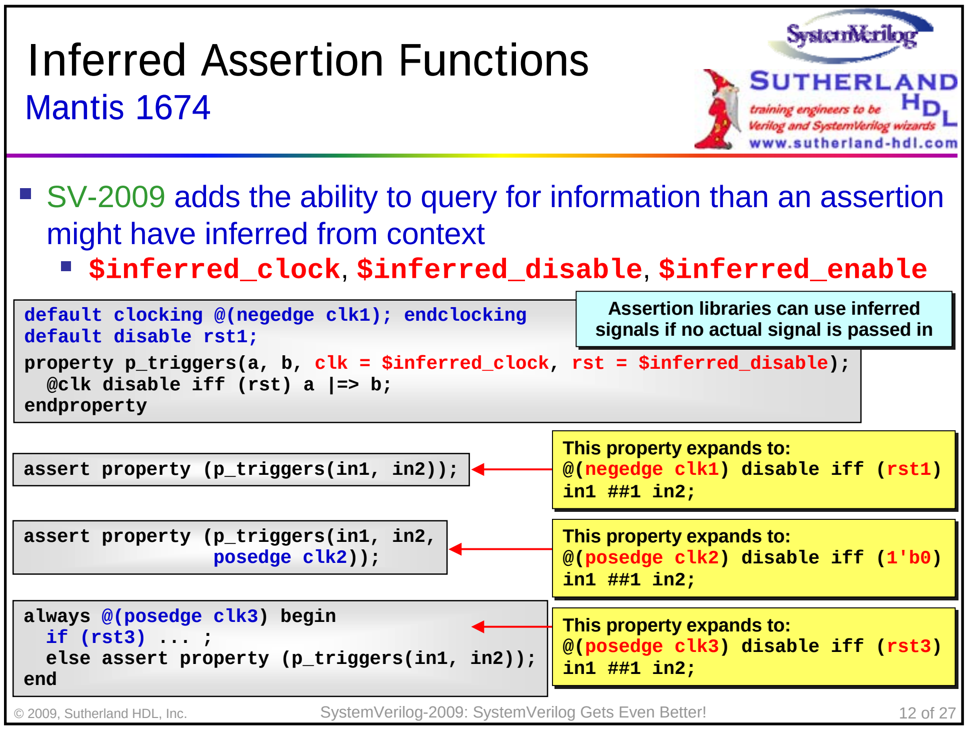
Assertion Past & Future Values
Mantis 1682

- SV-2009 adds functions that return the nearest past or future value of a signal as sampled by the global clock
  - One clock cycle in the past
    - **`$past_gclk`**, **`$rose_gclk`**, **`$fell_gclk`**, **`$stable_gclk`**, **`$changed_gclk`**
  - One clock cycle in the future
    - **`$future_gclk`**, **`$rising_gclk`**, **`$falling_gclk`**, **`$steady_gclk`**, **`$changing_gclk`**

```
assert property (
  @$global_clock
  $changing_gclk(data) |-> $falling_gclk(clock)
) else $error("data is not stable");
```

Verify that data only changes on a falling edge of clock

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 11 of 27



Inferred Assertion Functions
Mantis 1674

- SV-2009 adds the ability to query for information than an assertion might have inferred from context
  - **`$inferred_clock`**, **`$inferred_disable`**, **`$inferred_enable`**

```
default clocking @(negedge clk1); endclocking
default disable rst1;
property p_triggers(a, b, clk = $inferred_clock, rst = $inferred_disable);
  @clk disable iff (rst) a |=> b;
endproperty
```

Assertion libraries can use inferred signals if no actual signal is passed in

```
assert property (p_triggers(in1, in2));
```

This property expands to:
```
@(negedge clk1) disable iff (rst1)
in1 ##1 in2;
```

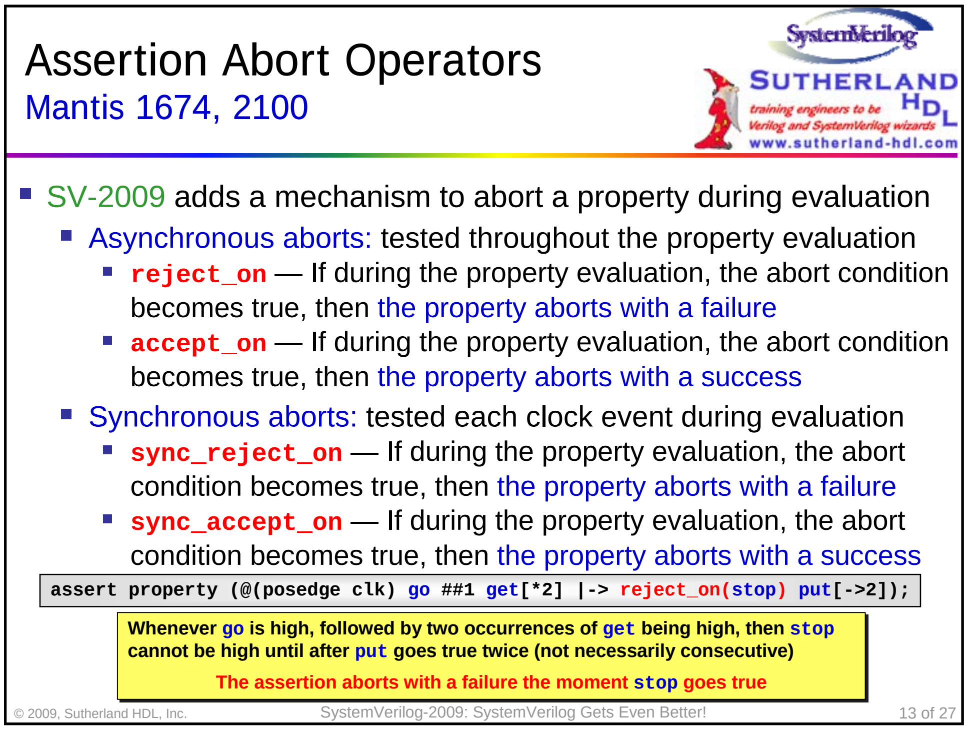
```
assert property (p_triggers(in1, in2,
                            posedge clk2));
```

This property expands to:
```
@(posedge clk2) disable iff (1'b0)
in1 ##1 in2;
```

```
always @(posedge clk3) begin
  if (rst3) ... ;
  else assert property (p_triggers(in1, in2));
end
```

This property expands to:
```
@(posedge clk3) disable iff (rst3)
in1 ##1 in2;
```

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 12 of 27



Assertion Abort Operators

Mantis 1674, 2100

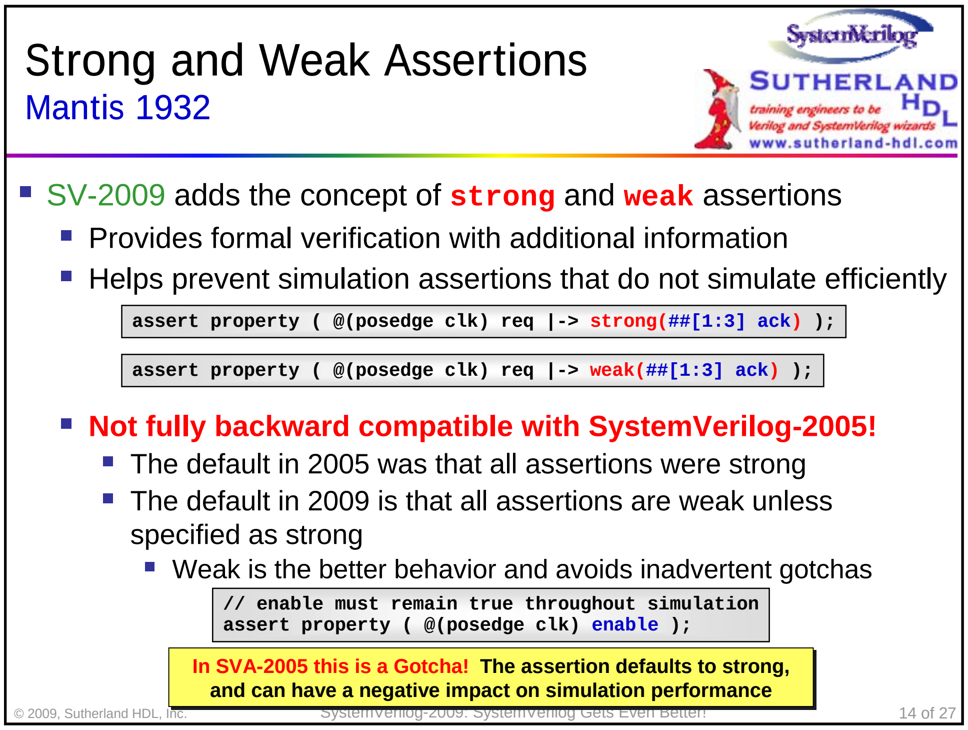
- SV-2009 adds a mechanism to abort a property during evaluation
  - Asynchronous aborts: tested throughout the property evaluation
    - **`reject_on`** — If during the property evaluation, the abort condition becomes true, then the property aborts with a failure
    - **`accept_on`** — If during the property evaluation, the abort condition becomes true, then the property aborts with a success
  - Synchronous aborts: tested each clock event during evaluation
    - **`sync_reject_on`** — If during the property evaluation, the abort condition becomes true, then the property aborts with a failure
    - **`sync_accept_on`** — If during the property evaluation, the abort condition becomes true, then the property aborts with a success

```
assert property (@(posedge clk) go ##1 get[*2] |-> reject_on(stop) put[->2]);
```

Whenever `go` is high, followed by two occurrences of `get` being high, then `stop` cannot be high until after `put` goes true twice (not necessarily consecutive)

The assertion aborts with a failure the moment `stop` goes true

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 13 of 27



Strong and Weak Assertions

Mantis 1932

- SV-2009 adds the concept of **`strong`** and **`weak`** assertions
  - Provides formal verification with additional information
  - Helps prevent simulation assertions that do not simulate efficiently

```
assert property ( @(posedge clk) req |-> strong(##[1:3] ack) );
```

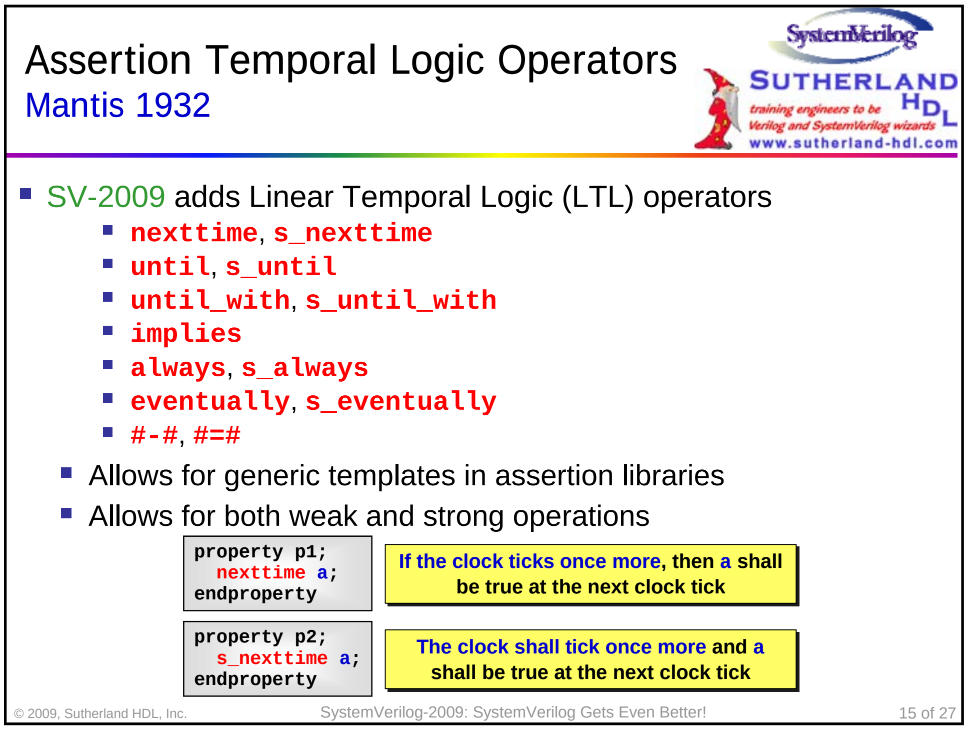
```
assert property ( @(posedge clk) req |-> weak(##[1:3] ack) );
```

  - **Not fully backward compatible with SystemVerilog-2005!**
    - The default in 2005 was that all assertions were strong
    - The default in 2009 is that all assertions are weak unless specified as strong
      - Weak is the better behavior and avoids inadvertent gotchas

```
// enable must remain true throughout simulation
assert property ( @(posedge clk) enable );
```

In SVA-2005 this is a Gotcha! The assertion defaults to strong, and can have a negative impact on simulation performance

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 14 of 27



Assertion Temporal Logic Operators
Mantis 1932

- SV-2009 adds Linear Temporal Logic (LTL) operators
  - `nexttime`, `s_nexttime`
  - `until`, `s_until`
  - `until_with`, `s_until_with`
  - `implies`
  - `always`, `s_always`
  - `eventually`, `s_eventually`
  - `#-#`, `#=#`
  - Allows for generic templates in assertion libraries
  - Allows for both weak and strong operations

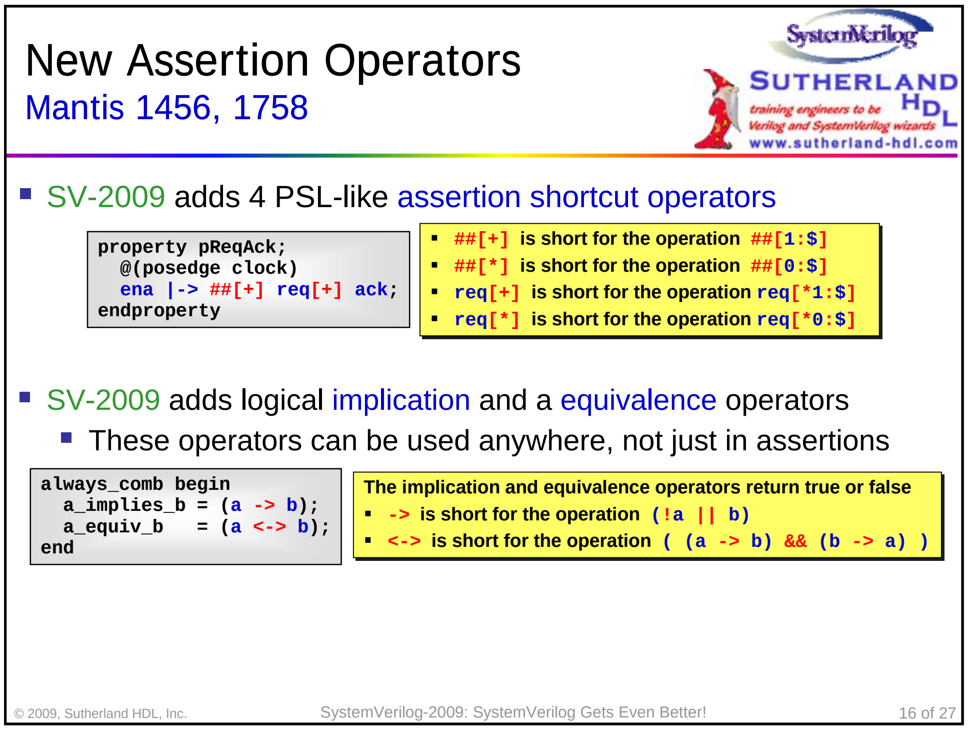
```
property p1;
  nexttime a;
endproperty
```

If the clock ticks once more, then a shall be true at the next clock tick

```
property p2;
  s_nexttime a;
endproperty
```

The clock shall tick once more and a shall be true at the next clock tick

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 15 of 27



New Assertion Operators
Mantis 1456, 1758

- SV-2009 adds 4 PSL-like assertion shortcut operators

```
property pReqAck;
  @(posedge clock)
  ena |-> ##[+] req[+] ack;
endproperty
```

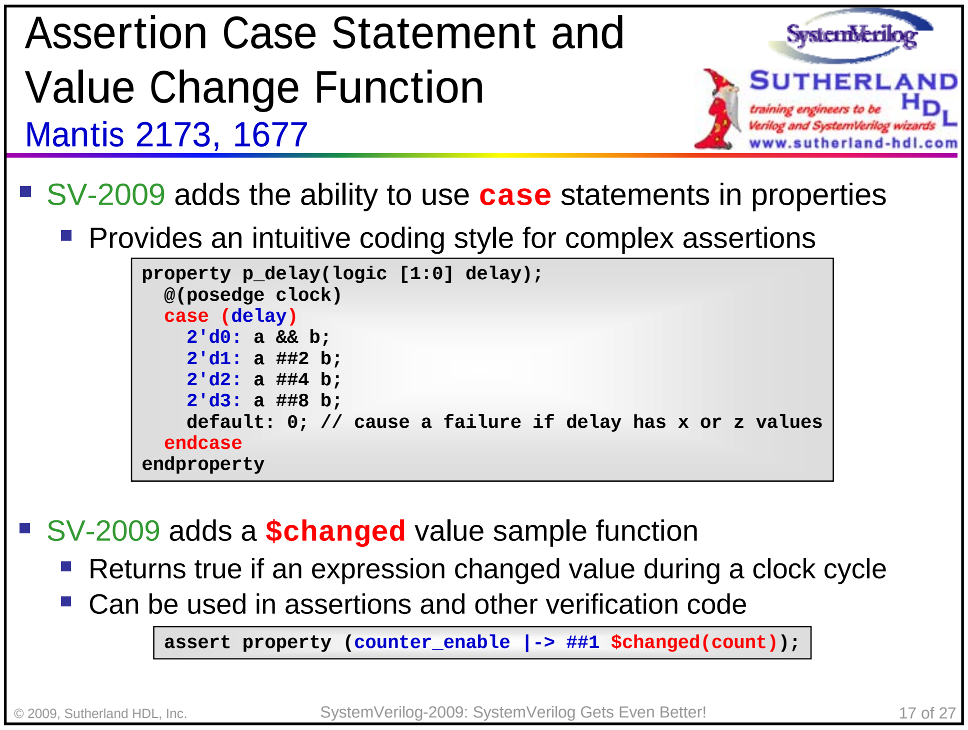
- `##[+]` **is short for the operation** `##[1:$]`
- `##[*]` **is short for the operation** `##[0:$]`
- `req[+]` **is short for the operation** `req[*1:$]`
- `req[*]` **is short for the operation** `req[*0:$]`

- SV-2009 adds logical implication and a equivalence operators
  - These operators can be used anywhere, not just in assertions

```
always_comb begin
  a_implies_b = (a -> b);
  a_equiv_b   = (a <-> b);
end
```

The implication and equivalence operators return true or false
- `->` **is short for the operation** `(!a || b)`
- `<->` **is short for the operation** `( (a -> b) && (b -> a) )`

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 16 of 27



Assertion Case Statement and Value Change Function
Mantis 2173, 1677

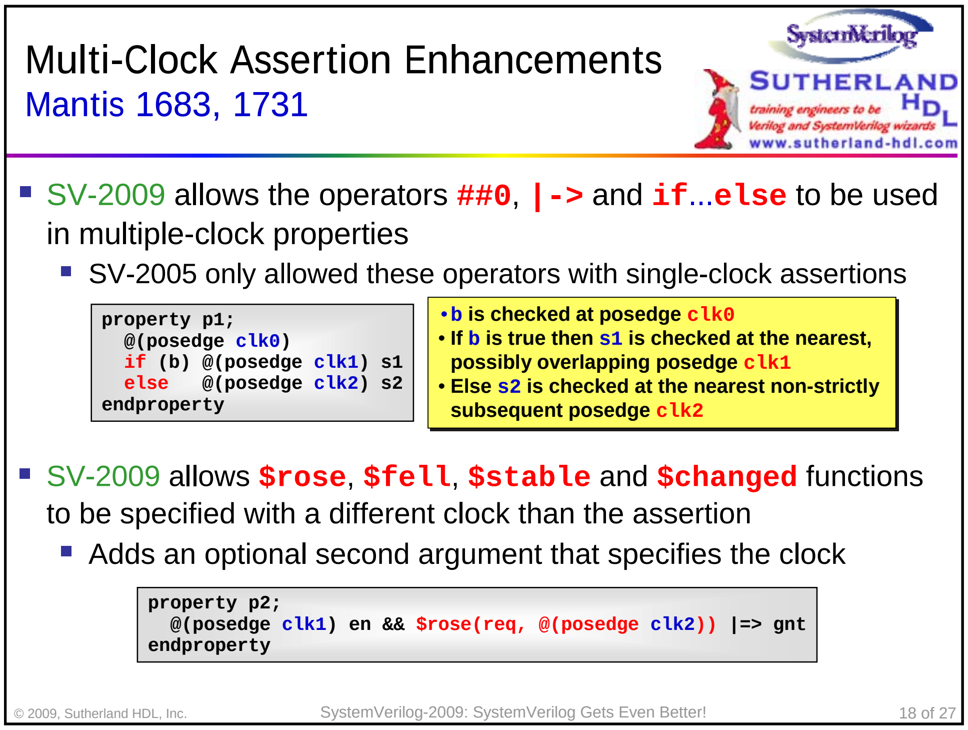
- SV-2009 adds the ability to use **case** statements in properties
  - Provides an intuitive coding style for complex assertions

```
property p_delay(logic [1:0] delay);
  @(posedge clock)
  case (delay)
    2'd0: a && b;
    2'd1: a ##2 b;
    2'd2: a ##4 b;
    2'd3: a ##8 b;
    default: 0; // cause a failure if delay has x or z values
  endcase
endproperty
```

- SV-2009 adds a **$changed** value sample function
  - Returns true if an expression changed value during a clock cycle
  - Can be used in assertions and other verification code

```
assert property (counter_enable |-> ##1 $changed(count));
```

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 17 of 27



Multi-Clock Assertion Enhancements
Mantis 1683, 1731

- SV-2009 allows the operators **##0**, **|->** and **if...else** to be used in multiple-clock properties
  - SV-2005 only allowed these operators with single-clock assertions

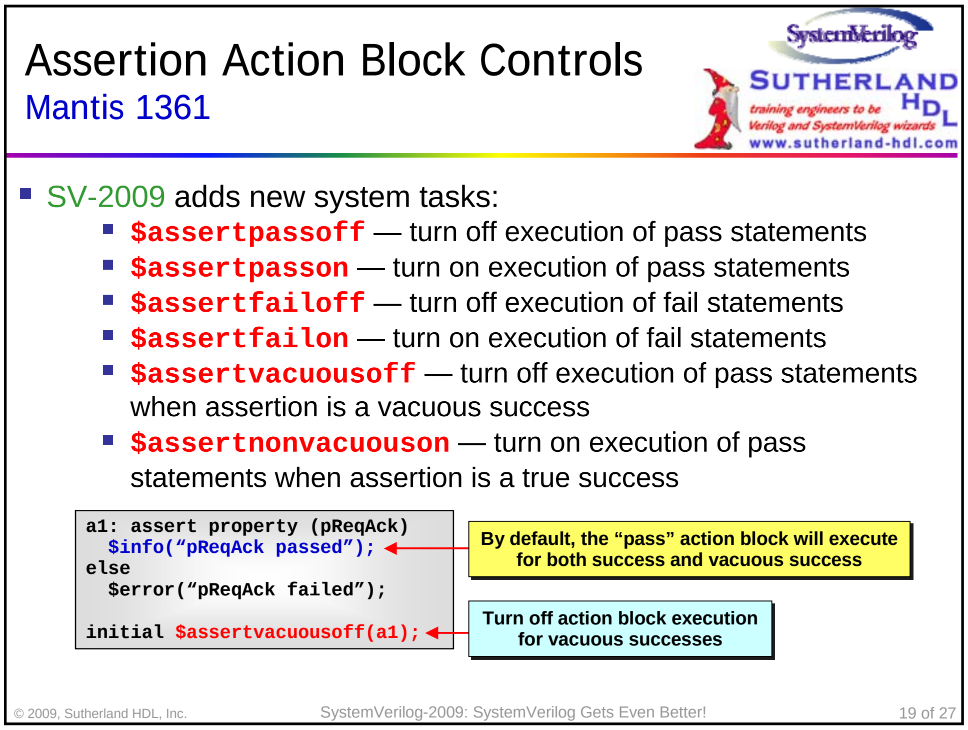
```
property p1;
  @(posedge clk0)
  if (b) @(posedge clk1) s1
  else   @(posedge clk2) s2
endproperty
```

- **b is checked at posedge clk0**
- **If b is true then s1 is checked at the nearest, possibly overlapping posedge clk1**
- **Else s2 is checked at the nearest non-strictly subsequent posedge clk2**

- SV-2009 allows **$rose**, **$fell**, **$stable** and **$changed** functions to be specified with a different clock than the assertion
  - Adds an optional second argument that specifies the clock

```
property p2;
  @(posedge clk1) en && $rose(req, @(posedge clk2)) |=> gnt
endproperty
```

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 18 of 27



Assertion Action Block Controls
Mantis 1361

- SV-2009 adds new system tasks:
  - $assertpassoff — turn off execution of pass statements
  - $assertpasson — turn on execution of pass statements
  - $assertfailoff — turn off execution of fail statements
  - $assertfailon — turn on execution of fail statements
  - $assertvacuousoff — turn off execution of pass statements when assertion is a vacuous success
  - $assertnonvacuouson — turn on execution of pass statements when assertion is a true success

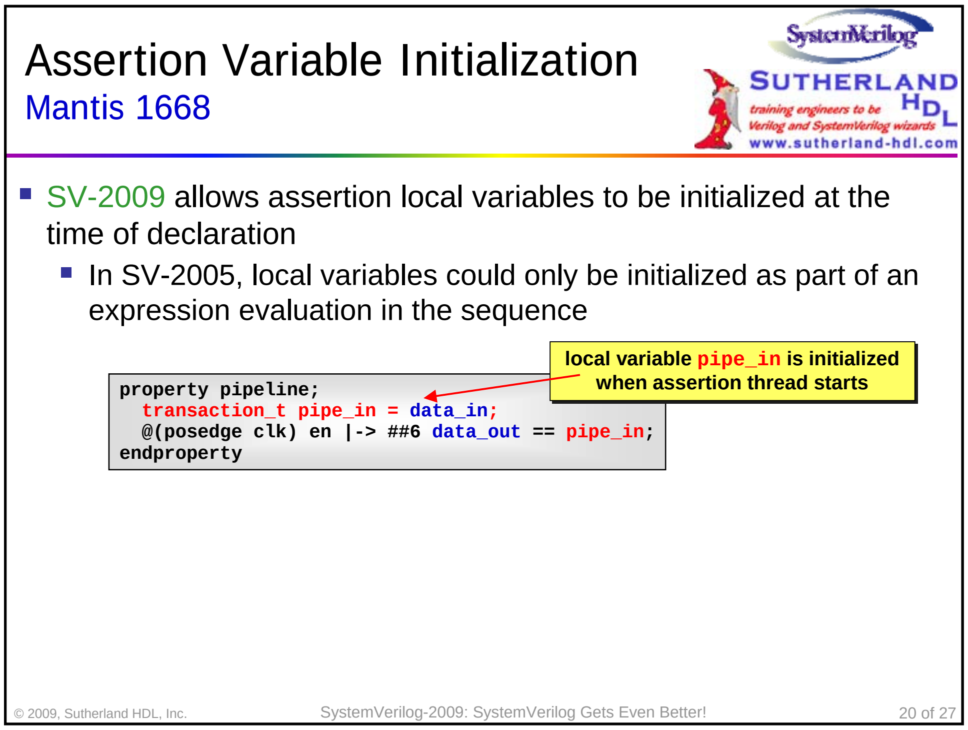
```
a1: assert property (pReqAck)
  $info("pReqAck passed");
else
  $error("pReqAck failed");

initial $assertvacuousoff(a1);
```

By default, the "pass" action block will execute for both success and vacuous success

Turn off action block execution for vacuous successes

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 19 of 27



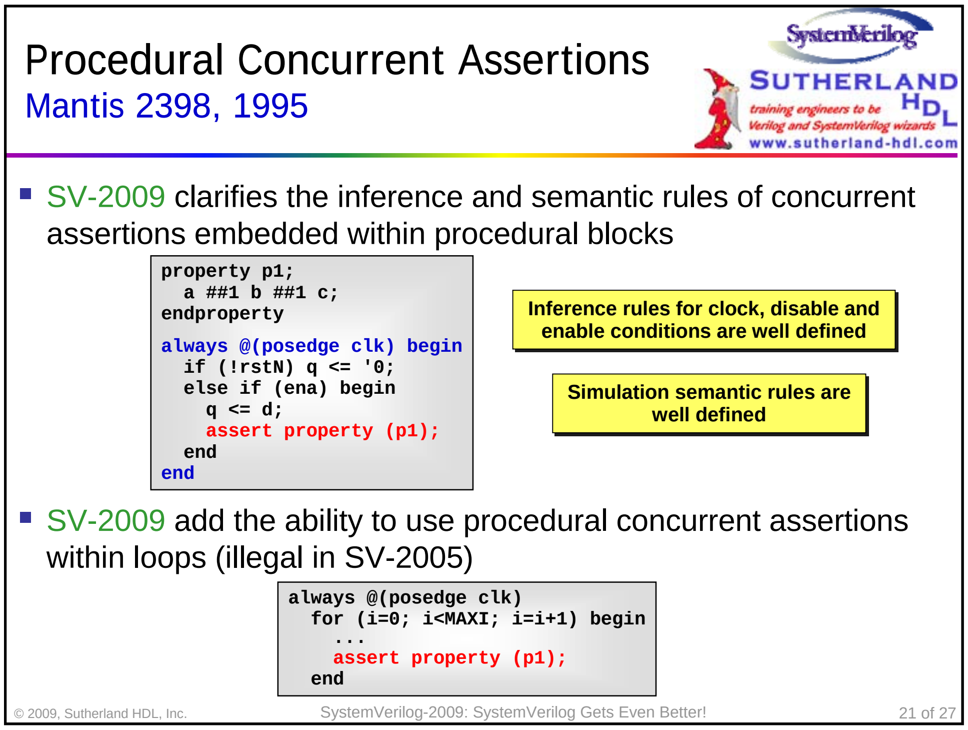
Assertion Variable Initialization
Mantis 1668

- SV-2009 allows assertion local variables to be initialized at the time of declaration
  - In SV-2005, local variables could only be initialized as part of an expression evaluation in the sequence

```
property pipeline;
  transaction_t pipe_in = data_in;
  @(posedge clk) en |-> ##6 data_out == pipe_in;
endproperty
```

local variable pipe_in is initialized when assertion thread starts

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 20 of 27



Procedural Concurrent Assertions
Mantis 2398, 1995

- SV-2009 clarifies the inference and semantic rules of concurrent assertions embedded within procedural blocks

```
property p1;
  a ##1 b ##1 c;
endproperty

always @(posedge clk) begin
  if (!rstN) q <= '0;
  else if (ena) begin
    q <= d;
    assert property (p1);
  end
end
```

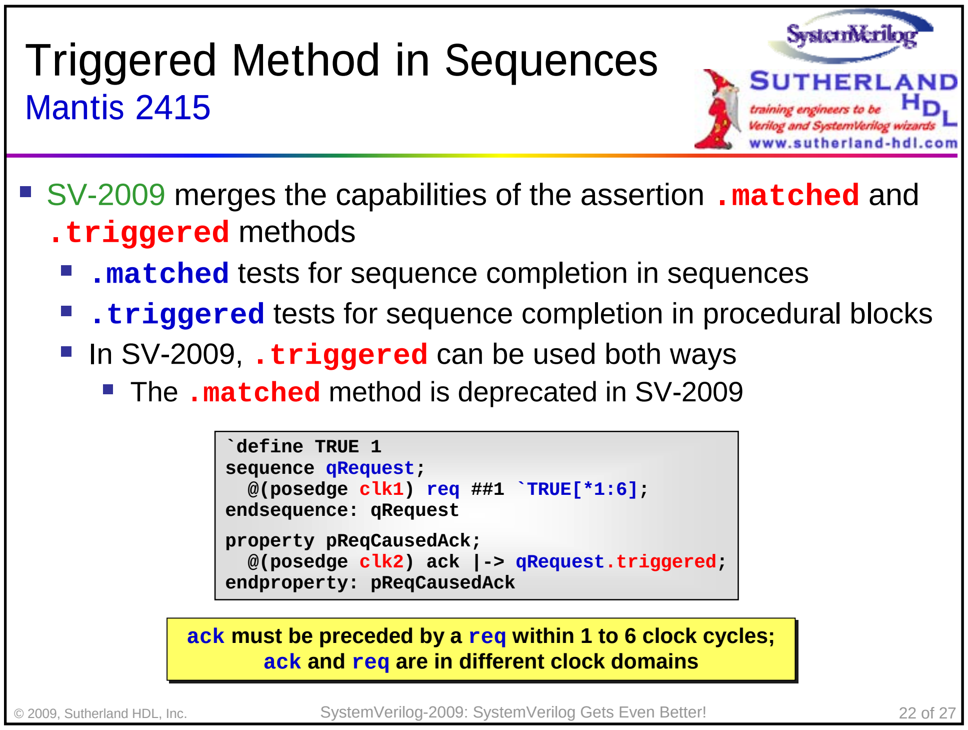
Inference rules for clock, disable and enable conditions are well defined

Simulation semantic rules are well defined

- SV-2009 add the ability to use procedural concurrent assertions within loops (illegal in SV-2005)

```
always @(posedge clk)
  for (i=0; i<MAXI; i=i+1) begin
    ...
    assert property (p1);
  end
```

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 21 of 27



Triggered Method in Sequences
Mantis 2415

- SV-2009 merges the capabilities of the assertion `.matched` and `.triggered` methods
  - `.matched` tests for sequence completion in sequences
  - `.triggered` tests for sequence completion in procedural blocks
  - In SV-2009, `.triggered` can be used both ways
    - The `.matched` method is deprecated in SV-2009

```
`define TRUE 1
sequence qRequest;
  @(posedge clk1) req ##1 `TRUE[*1:6];
endsequence: qRequest

property pReqCausedAck;
  @(posedge clk2) ack |-> qRequest.triggered;
endproperty: pReqCausedAck
```

ack must be preceded by a req within 1 to 6 clock cycles; ack and req are in different clock domains

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 22 of 27

Assume, Cover and Restrict Assertion Statements
Mantis 1460, 1726, 1806

- SV-2009 adds action blocks to assume statements
    - `assume property (property_spec) action_block`
  - Makes the syntax for assume statements consistent with concurrent assert and cover statements
- SV-2009 adds immediate `cover` and `assume` statements
    - `cover (expression) action_block`
    - `assume (expression) action_block`
  - Makes the syntax for assume statements consistent with assert and cover statements
- SV-2009 adds a `restrict` statements
    - `restrict property (property_spec);`
  - Restricts the value state space used by formal verification tools
  - Ignored by simulators

```
restrict property (@(posedge clk) mode == 0);
```

Deferred Immediate Assertions
Mantis 2005

- SV-2009 adds "*deferred immediate assertions*"
    - `assert #0 (expression) action_block`
    - `assume #0 (expression) action_block`
    - `cover  #0 (expression) action_block`
  - Only the last assertion evaluation is used if the assertion is executed multiple times in the same time step
  - Prevents erroneous assertion messages

```
assign not_a = !a;

always_comb begin
  a1: assert (not_a != a);
  a2: assert #0 (not_a != a);
end
```

Simple immediate assertion could trigger on a glitch as a and not_a change values

Deferred immediate assertion evaluates after a and not_a have stabilized in a time step

Assertions at Elaboration Time

Mantis 1769

- SV-2009 adds the ability to print assertion severity messages at elaboration time, before simulation starts running
  - When `$fatal`, `$error`, `$warning` or `$info` are used outside of an assertion or procedural block, they are executed at elaboration
    - Can be called as a stand-alone statement
    - Can be called from within a generate block
  - Can be used to check the validity of parameter definitions

```
module my_chip #(parameter N = 1)
                (input  [N-1:0] in,
                 output [N-1:0] out);
  generate
    if ((N < 1) || (N > 8))
      $fatal(1, "Parameter N has an invalid value of %0d", N);
  endgenerate
  ...
endmodule
```

New Keywords

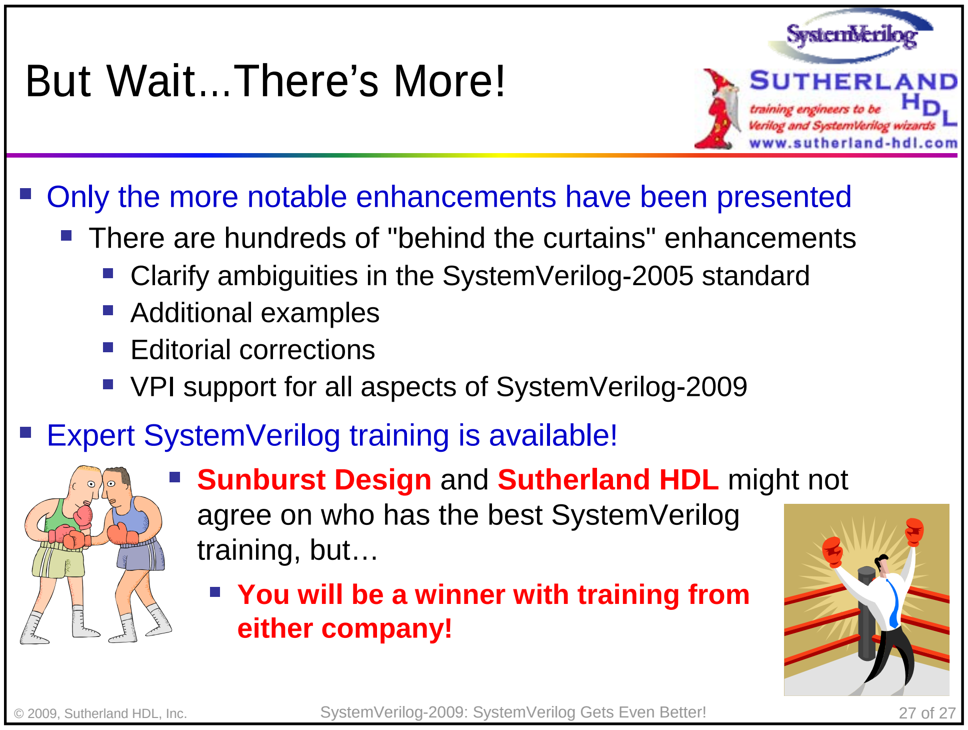
- SystemVerilog-2009 reserves several additional keywords

accept_on	let	s_nexttime	unique0
checker	nexttime	s_until	until
endchecker	reject_on	s_until_with	until_with
eventually	restrict	strong	untyped
global	s_always	sync_accept_on	weak
implies	s_eventually	sync_reject_on	

- SV-2009 adds a new argument to `begin_keywords
  - Maintains keyword backward compatibility with previous versions of the Verilog and SystemVerilog standard

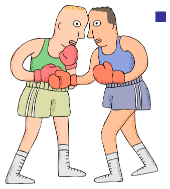
```
`begin_keywords "1800-2005"
module old_chip (...);
  ...
endmodule
`end_keywords
```

```
`begin_keywords "1800-2009"
module new_chip (...);
  ...
endmodule
`end_keywords
```



But Wait...There's More!

- Only the more notable enhancements have been presented
  - There are hundreds of "behind the curtains" enhancements
    - Clarify ambiguities in the SystemVerilog-2005 standard
    - Additional examples
    - Editorial corrections
    - VPI support for all aspects of SystemVerilog-2009
- Expert SystemVerilog training is available!



  - **Sunburst Design** and **Sutherland HDL** might not agree on who has the best SystemVerilog training, but…
    - **You will be a winner with training from either company!**

© 2009, Sutherland HDL, Inc. SystemVerilog-2009: SystemVerilog Gets Even Better! 27 of 27