A Summary of Changes in the Proposed SystemVerilog 2012 Standard

by Sutherland HDL, Inc., Portland, Oregon © 2012
www.sutherland-hdl.com

DAC – 2012

SystemVerilog Birds of a Feather

A Summary of Changes in the Proposed System Verilog-2012 Standard

by Stuart Sutherland
Sutherland HDL
Training Engineers to be SystemVerilog wizards
www.sutherland-hdl.com

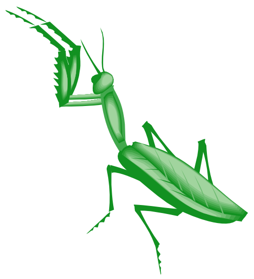
Verilog/SystemVerilog Evolution

- SystemVerilog is continuing to evolve as the complexity of design and verification evolves
- The proposed SystemVerilog-2012 standard includes…
  - **31 new features** added to the language
  - 60 clarifications to existing language features
  - 71 corrections (typos, English grammar, punctuation, etc.)
  - Dozens of minor editorial corrections (font usage, punctuation)

The focus of this presentation is on the 31 new language features, and how those features can help make writing complex verification testbenches simpler or more efficient

Finding the Full Details

- All changes considered for the IEEE standard are tracked in an online data base called "Mantis"
  - The enhancements on the following slides contain a "mantis number"
    - The data base entry for that number contains the details for each new SystemVerilog feature
  - The online data base can be accessed at:



www.verilog.org/svdb

- **Click on the Login link**
- **Username: guest**
- **Password: guest**
- **In the Projects box, select SystemVerilog P1800**

Typed new() Constructors

Mantis 3001

- Before…
  - The object handle type and the new() type must be identical
  - To create a child object and assign to a parent handle took 3 steps

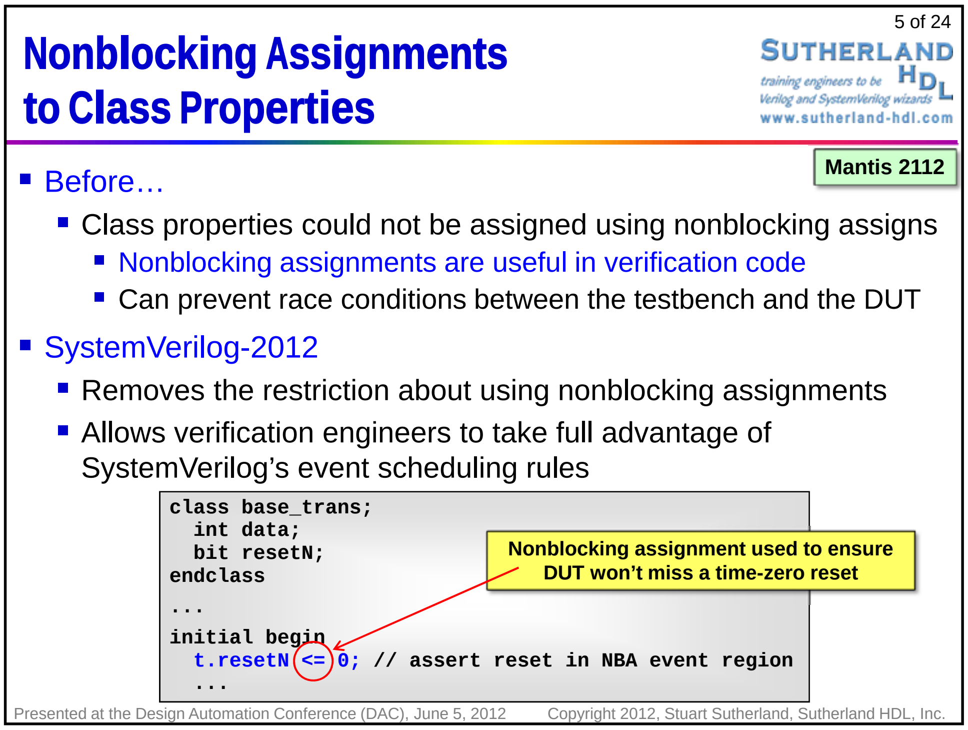
```
class base_trans; ... endclass

class reset_trans extends base_trans;... endclass

base_trans  t_base;
reset_trans t_reset = new;
t_base = t_reset;
```

multiple assignments are required to set the t_base handle

- SystemVerilog-2012
  - The call to new() can be "typed" using its class name
    - The return must be assigned to a handle of the same class type or a parent/grandparent of that type

```
base_trans t_base = reset_trans::new;
```

- **Fewer lines of code**
- **Self-documenting code**
- **Less risk of obscure errors**



Nonblocking Assignments to Class Properties

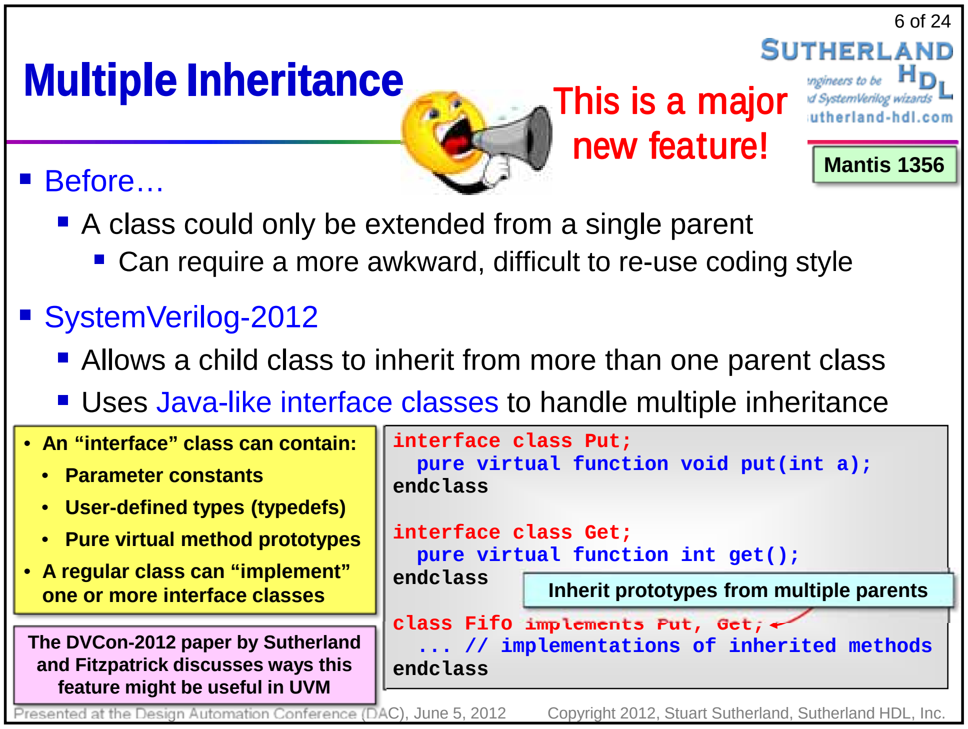
SUTHERLAND HDL
training engineers to be Verilog and SystemVerilog wizards
www.sutherland-hdl.com

Mantis 2112

- Before…
  - Class properties could not be assigned using nonblocking assigns
    - Nonblocking assignments are useful in verification code
    - Can prevent race conditions between the testbench and the DUT
- SystemVerilog-2012
  - Removes the restriction about using nonblocking assignments
  - Allows verification engineers to take full advantage of SystemVerilog's event scheduling rules

```
class base_trans;
  int data;
  bit resetN;
endclass
...
initial begin
  t.resetN <= 0; // assert reset in NBA event region
  ...
```

Nonblocking assignment used to ensure DUT won't miss a time-zero reset



Multiple Inheritance

This is a major new feature!

SUTHERLAND HDL
training engineers to be Verilog and SystemVerilog wizards
www.sutherland-hdl.com

Mantis 1356

- Before…
  - A class could only be extended from a single parent
    - Can require a more awkward, difficult to re-use coding style
- SystemVerilog-2012
  - Allows a child class to inherit from more than one parent class
  - Uses Java-like interface classes to handle multiple inheritance

- An "interface" class can contain:
  - Parameter constants
  - User-defined types (typedefs)
  - Pure virtual method prototypes
- A regular class can "implement" one or more interface classes

The DVCon-2012 paper by Sutherland and Fitzpatrick discusses ways this feature might be useful in UVM

```
interface class Put;
  pure virtual function void put(int a);
endclass

interface class Get;
  pure virtual function int get();
endclass

class Fifo implements Put, Get;
  ... // implementations of inherited methods
endclass
```

Inherit prototypes from multiple parents

Soft Constraints

SUTHERLAND HDL
training engineers to be Verilog and SystemVerilog wizards
www.sutherland-hdl.com

Mantis 2987

- Before… **This is also a big enhancement!**
  - All randomization constraints were “hard” constraints
    - An error results if a constraint conflicts with another constraint

Example:
- **A transaction class has constraints, but a specific test requires a different constraint**
  - **An error will occur if the specific constraint conflicts with the built-in constraint**
- **The verification engineer writing the test must write extra code to avoid potential conflicts**

- SystemVerilog-2012
  - Constraints can be specified as “soft”
    - Ignored if conflicts with another constraint

```
class Packet;
  rand int pkt_size;
  constraint size {soft pkt_size inside {32,1024};}
endclass

Packet p = new();
p.randomize with {pkt_size == 512;}
```

The built-in constraint is that pkt_size will have a random value of either 32 or 1024, only

The randomize with() constraint takes precedence over the soft constraint, instead of resulting in a run-time error

Uniqueness Constraints

SUTHERLAND HDL
training engineers to be Verilog and SystemVerilog wizards
www.sutherland-hdl.com

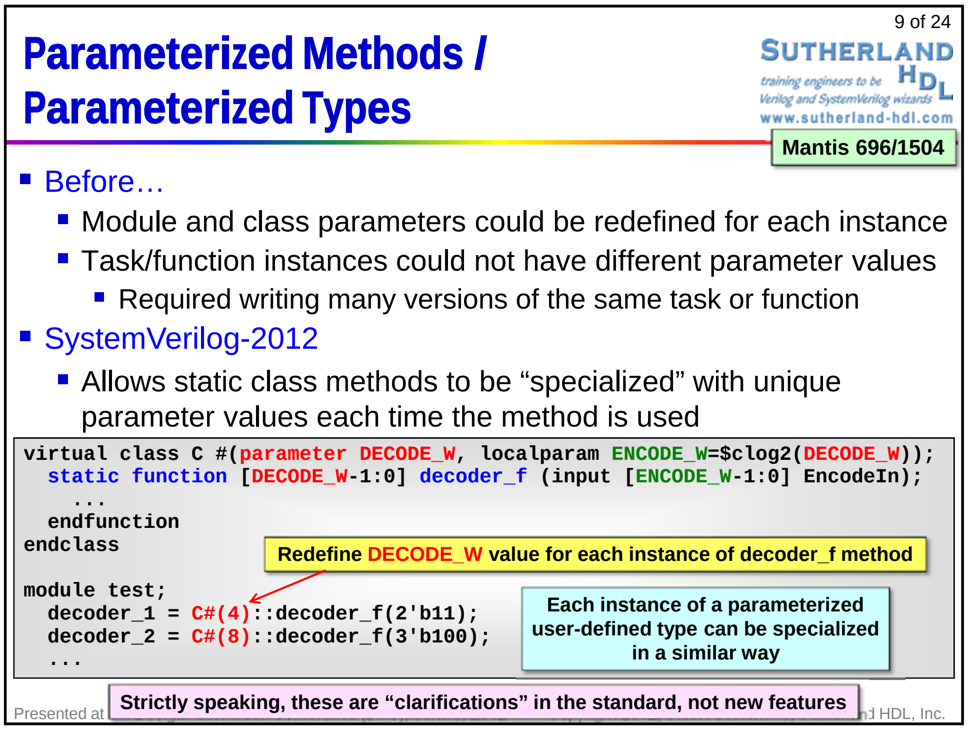
Mantis 3028

- Before…
  - There was no simple way to specify constraints so that several variables — or all the members of an array — had different random values and none had the same value
- SystemVerilog-2012
  - Adds a uniqueness constraint that where all variables in a list or an array receive unique values

```
class Transaction;
  rand int  a, b, c;
  rand byte data_array[16];

  constraint c1 { unique {a,b,c}; }
  constraint c2 { unique {data_array}; }
endclass
```

Constraint c1 ensures that, when random values are generated, the values of a, b and c will be different

Constraint c2 ensures that when random values are generated, every element of data_array will have a different value



Parameterized Methods / Parameterized Types

Mantis 696/1504

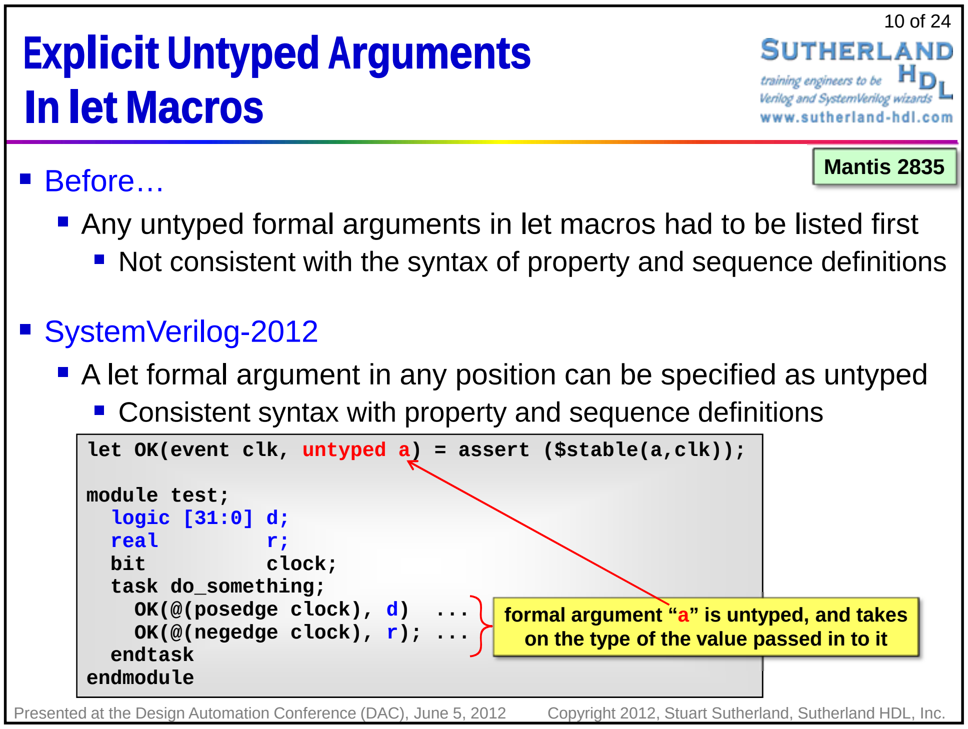
- Before…
  - Module and class parameters could be redefined for each instance
  - Task/function instances could not have different parameter values
    - Required writing many versions of the same task or function
- SystemVerilog-2012
  - Allows static class methods to be "specialized" with unique parameter values each time the method is used

```
virtual class C #(parameter DECODE_W, localparam ENCODE_W=$clog2(DECODE_W));
  static function [DECODE_W-1:0] decoder_f (input [ENCODE_W-1:0] EncodeIn);
    ...
  endfunction
endclass

module test;
  decoder_1 = C#(4)::decoder_f(2'b11);
  decoder_2 = C#(8)::decoder_f(3'b100);
  ...
```

Redefine DECODE_W value for each instance of decoder_f method

Each instance of a parameterized user-defined type can be specialized in a similar way

Strictly speaking, these are "clarifications" in the standard, not new features



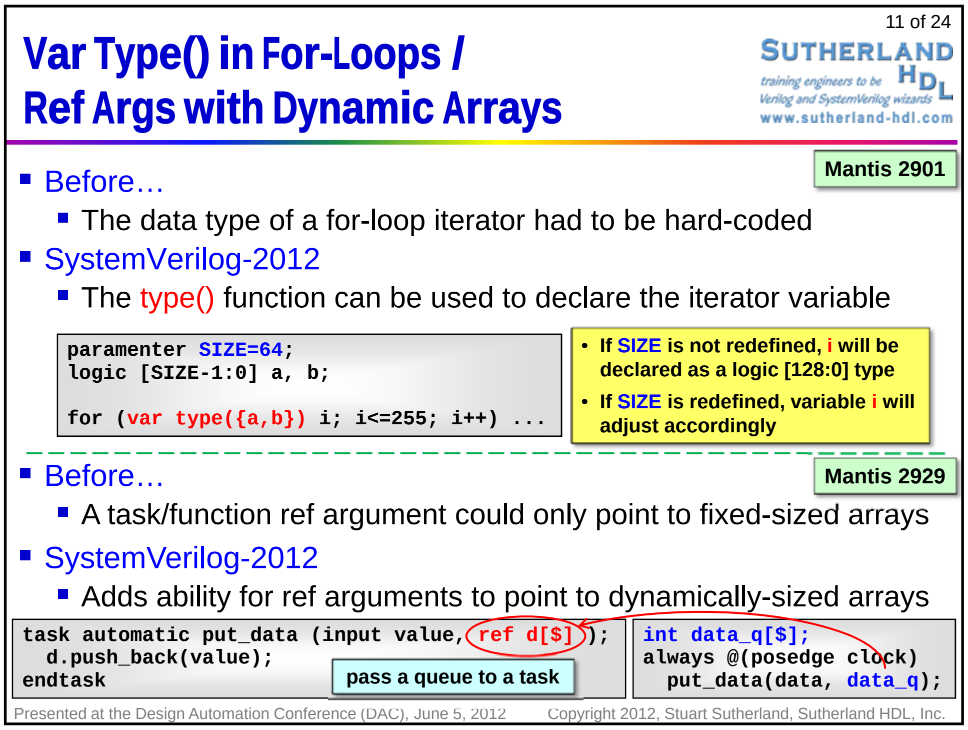
Explicit Untyped Arguments In let Macros

Mantis 2835

- Before…
  - Any untyped formal arguments in let macros had to be listed first
    - Not consistent with the syntax of property and sequence definitions
- SystemVerilog-2012
  - A let formal argument in any position can be specified as untyped
    - Consistent syntax with property and sequence definitions

```
let OK(event clk, untyped a) = assert ($stable(a,clk));

module test;
  logic [31:0] d;
  real         r;
  bit          clock;
  task do_something;
    OK(@(posedge clock), d)  ...
    OK(@(negedge clock), r); ...
  endtask
endmodule
```

formal argument "a" is untyped, and takes on the type of the value passed in to it



Var Type() in For-Loops / Ref Args with Dynamic Arrays

Mantis 2901

- Before…
  - The data type of a for-loop iterator had to be hard-coded
- SystemVerilog-2012
  - The type() function can be used to declare the iterator variable

```
parementer SIZE=64;
logic [SIZE-1:0] a, b;

for (var type({a,b}) i; i<=255; i++) ...
```

- If SIZE is not redefined, i will be declared as a logic [128:0] type
- If SIZE is redefined, variable i will adjust accordingly

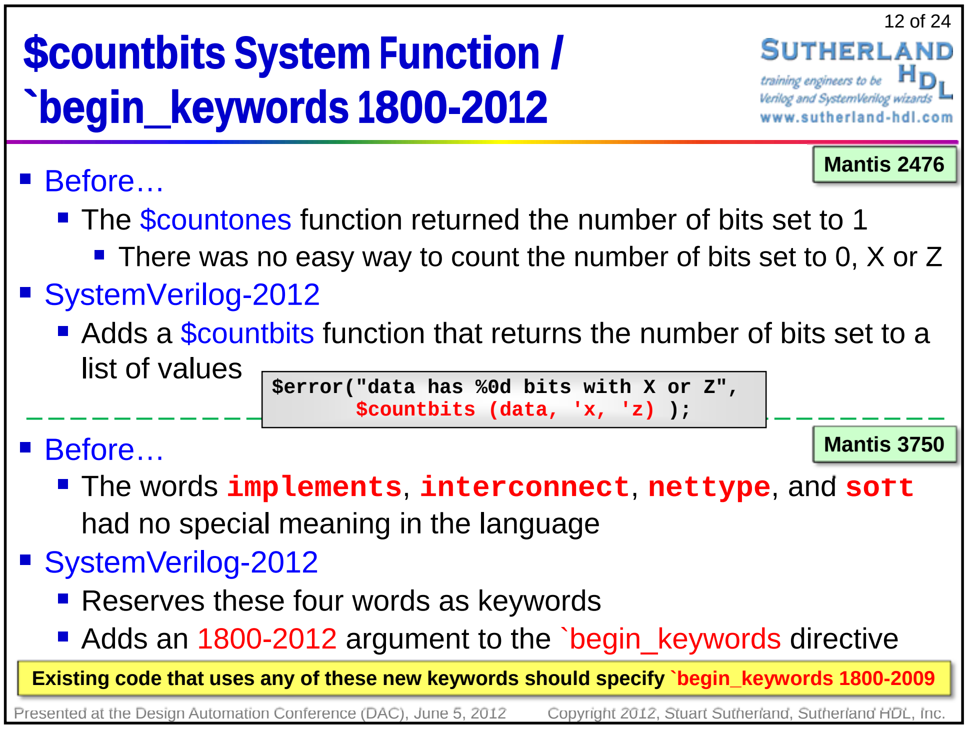
Mantis 2929

- Before…
  - A task/function ref argument could only point to fixed-sized arrays
- SystemVerilog-2012
  - Adds ability for ref arguments to point to dynamically-sized arrays

```
task automatic put_data (input value, ref d[$]);
  d.push_back(value);
endtask
```

pass a queue to a task

```
int data_q[$];
always @(posedge clock)
  put_data(data, data_q);
```



$countbits System Function / `begin_keywords 1800-2012

Mantis 2476

- Before…
  - The $countones function returned the number of bits set to 1
    - There was no easy way to count the number of bits set to 0, X or Z
- SystemVerilog-2012
  - Adds a $countbits function that returns the number of bits set to a list of values

```
$error("data has %0d bits with X or Z",
       $countbits (data, 'x, 'z) );
```

Mantis 3750

- Before…
  - The words **implements**, **interconnect**, **nettype**, and **soft** had no special meaning in the language
- SystemVerilog-2012
  - Reserves these four words as keywords
  - Adds an 1800-2012 argument to the `begin_keywords directive

Existing code that uses any of these new keywords should specify `begin_keywords 1800-2009

User-Defined Net Types / Typeless Netlists

SUTHERLAND HDL
training engineers to be Verilog and SystemVerilog wizards
www.sutherland-hdl.com

These new features are important for mixed signal designs!

Mantis 3398

- Before…
  - Engineers could only create user-defined types based on variables
- SystemVerilog-2012
  - Adds ability to create user-defined net types based on net types
    - Can define custom nets for 2-state and floating point values
    - Can define custom resolution functions for multi-driver logic

Mantis 3724

- Before…
  - Netlists had to be hardcoded to only use specific net types
- SystemVerilog-2012
  - Adds a generic net that infers its type from lower-level connections
    - Enables using configurations to select design versions (e.g. digital or analog versions of a module) without modifying the netlist

Coverpoint Variables / bins...with() Construct / Coverage Functions

SUTHERLAND HDL
training engineers to be Verilog and SystemVerilog wizards
www.sutherland-hdl.com

Mantis 2506

These 3 enhancements can help improve coverage specs and goals!

- Before…
  - Coverpoint labels could not be used in expressions
  - Coverage expressions could not call functions
  - Coverage bins could not easily exclude specific values
- SystemVerilog-2012
  - Coverpoint labels are variables that can be used in expressions
  - Coverage expressions can call functions (eliminates duplicate code used by multiple coverpoints)
  - A bins...with() construct can be used to exclude values in a bin that would not be of interest in a test

```
a: coverpoint data {
  bins mod16[] = {[0:255]} with (item % 16 == 0);
}
```

mod16 only tracks values that are evenly divisible by 16

("item" is a variable that is built into bins...with())

Assertion Data Types / Sampled Value Data Types

Mantis 2328/2353

- Before…
  - Assertions were limited to testing simple, integral values
- SystemVerilog-2012
  - Assertions can now test real (floating point) values and dynamic arrays (such as strings and queues) and static class properties

```
byte q[$];
property p1;
  $rose(write) |-> q[0];
endproperty
```

A dynamic queue array

Mantis 3213

- Before…
  - Value sampling functions, such as $sample() were limited to testing simple, integral values
- SystemVerilog-2012
  - Value sampling functions can now test real values and dynamic arrays (e.g.: strings and queues) and static class properties

Global Clock Resolution

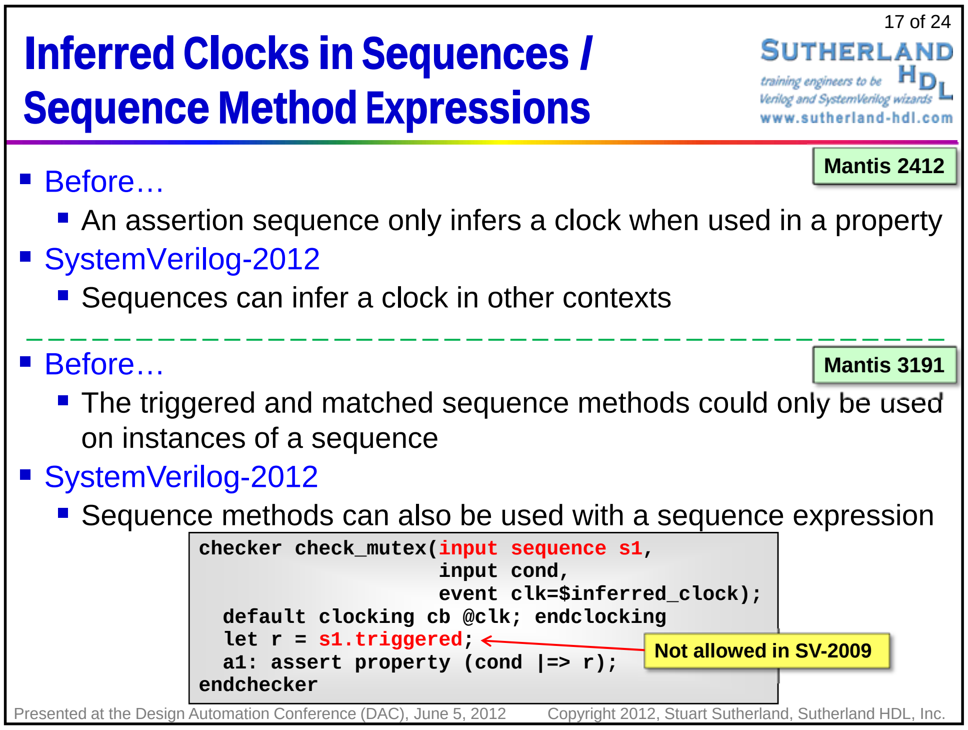
Mantis 3069

- Before…
  - There could only be a single global clock definition, which encompassed the entire design
    - Made it difficult to verify designs with multiple clock domains
- SystemVerilog-2012
  - Each hierarchy scope can have a different global clock
    - Applies to all sub-scopes until a new global clock is defined

```
module master (...);
  ...
  global clocking @(posedge m_clock);
  endclocking
  ...
  property @($global_clock)
    ...
    endproperty
  ...
endmodule
```

```
module slave (...);
  ...
  global clocking @(posedge s_clock);
  endclocking
  ...
  property @($global_clock)
    ...
  endproperty
  ...
endmodule
```

NOTE: This enhancement is not backward compatible with SystemVerilog-2009



Inferred Clocks in Sequences / Sequence Method Expressions

Mantis 2412

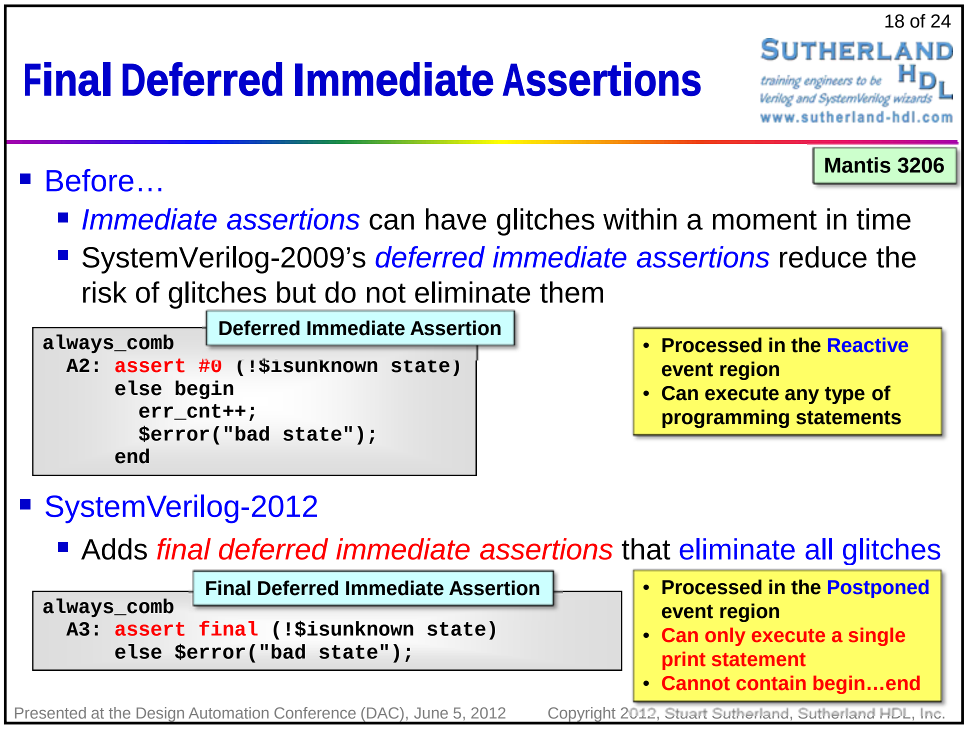
- Before…
  - An assertion sequence only infers a clock when used in a property
- SystemVerilog-2012
  - Sequences can infer a clock in other contexts

Mantis 3191

- Before…
  - The triggered and matched sequence methods could only be used on instances of a sequence
- SystemVerilog-2012
  - Sequence methods can also be used with a sequence expression

```
checker check_mutex(input sequence s1,
                    input cond,
                    event clk=$inferred_clock);
  default clocking cb @clk; endclocking
  let r = s1.triggered;
  a1: assert property (cond |=> r);
endchecker
```

Not allowed in SV-2009



Final Deferred Immediate Assertions

Mantis 3206

- Before…
  - *Immediate assertions* can have glitches within a moment in time
  - SystemVerilog-2009's *deferred immediate assertions* reduce the risk of glitches but do not eliminate them

Deferred Immediate Assertion

```
always_comb
  A2: assert #0 (!$isunknown state)
      else begin
        err_cnt++;
        $error("bad state");
      end
```

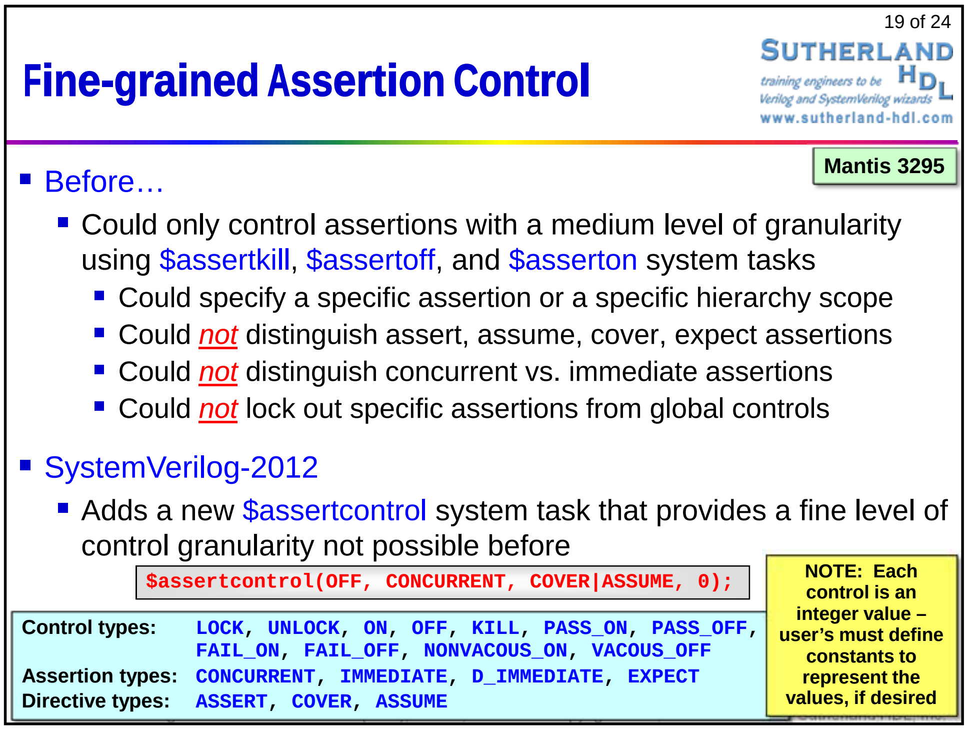
- Processed in the Reactive event region
- Can execute any type of programming statements

- SystemVerilog-2012
  - Adds *final deferred immediate assertions* that eliminate all glitches

Final Deferred Immediate Assertion

```
always_comb
  A3: assert final (!$isunknown state)
      else $error("bad state");
```

- Processed in the Postponed event region
- Can only execute a single print statement
- Cannot contain begin…end



Fine-grained Assertion Control

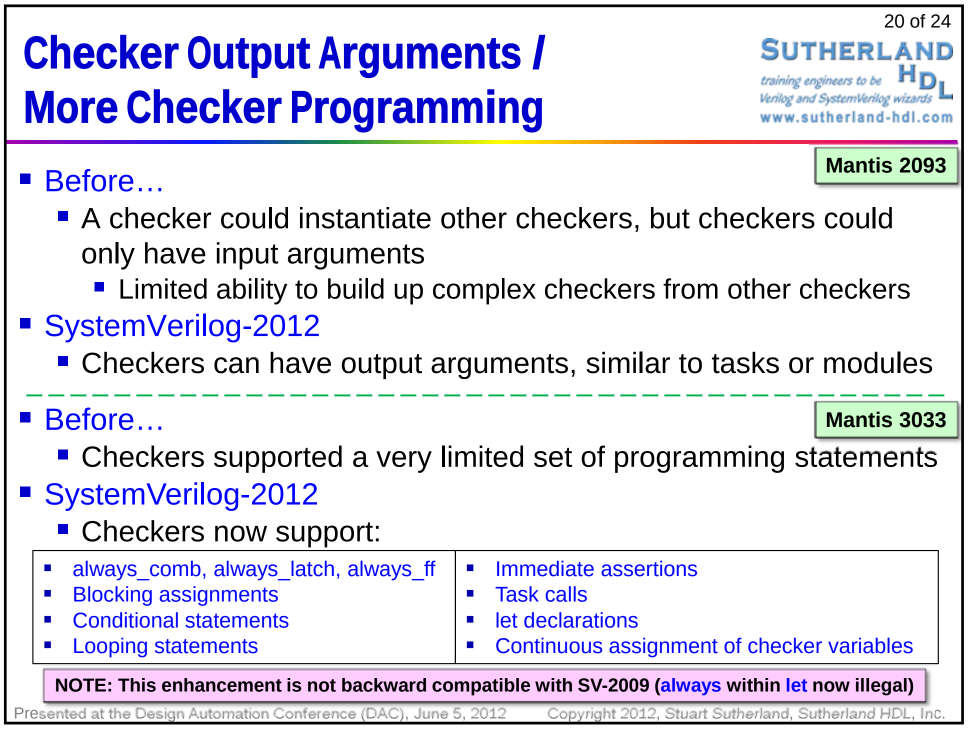
Mantis 3295

- Before…
  - Could only control assertions with a medium level of granularity using $assertkill, $assertoff, and $asserton system tasks
    - Could specify a specific assertion or a specific hierarchy scope
    - Could *not* distinguish assert, assume, cover, expect assertions
    - Could *not* distinguish concurrent vs. immediate assertions
    - Could *not* lock out specific assertions from global controls
- SystemVerilog-2012
  - Adds a new $assertcontrol system task that provides a fine level of control granularity not possible before

```
$assertcontrol(OFF, CONCURRENT, COVER|ASSUME, 0);
```

Control types:	`LOCK, UNLOCK, ON, OFF, KILL, PASS_ON, PASS_OFF, FAIL_ON, FAIL_OFF, NONVACOUS_ON, VACOUS_OFF`
Assertion types:	`CONCURRENT, IMMEDIATE, D_IMMEDIATE, EXPECT`
Directive types:	`ASSERT, COVER, ASSUME`

NOTE: Each control is an integer value – user's must define constants to represent the values, if desired



Checker Output Arguments / More Checker Programming

Mantis 2093

- Before…
  - A checker could instantiate other checkers, but checkers could only have input arguments
    - Limited ability to build up complex checkers from other checkers
- SystemVerilog-2012
  - Checkers can have output arguments, similar to tasks or modules

Mantis 3033

- Before…
  - Checkers supported a very limited set of programming statements
- SystemVerilog-2012
  - Checkers now support:

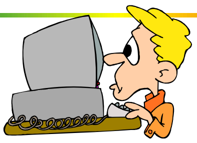
▪ always_comb, always_latch, always_ff	▪ Immediate assertions
▪ Blocking assignments	▪ Task calls
▪ Conditional statements	▪ let declarations
▪ Looping statements	▪ Continuous assignment of checker variables

NOTE: This enhancement is not backward compatible with SV-2009 (always within let now illegal)

VPI Enhancements

SUTHERLAND HDL
training engineers to be Verilog and SystemVerilog wizards
www.sutherland-hdl.com

Mantis 3116 Mantis 3193
Mantis 3188 Mantis 3884



- Before…
  - The SystemVerilog Verification Procedural Interface (VPI) supported constructs in the SystemVerilog-2009 standard
- SystemVerilog-2012
  - The VPI was enhanced to support the new features added in SystemVerilog-2012
    - VPI support for soft constraints
    - VPI access added to the built-in process class
    - VPI transition to typespecs added to named events
    - VPI join type property added to the Scope diagram
  - Many other minor enhancements and clarifications were made to the SystemVeriog-2012 VPI

Questions and Discussion...

SUTHERLAND HDL
training engineers to be Verilog and SystemVerilog wizards
www.sutherland-hdl.com

- In what ways will these 31 new features be useful in your projects?
  - OOP enhancements
    - Typed new() constructors
    - Nonblocking assignments
    - Multiple inheritance
  - Constrained random enhancements
    - Soft constraints
    - Uniqueness constraints
  - Programming enhancements
    - Parameterized tasks and functions
    - Parameterized user-defined types
    - Untyped arguments in let constructs
    - var type() in for-loops
    - ref arguments with dynamic arrays
    - $countbits system function
    - `begin_keywords 1800-2012
  - Mixed-signal enhancements
    - User-defined net types
    - Typeless netlist connections
  - Coverage enhancements
    - Coverpoint variables
    - bins…with() expressions
    - Coverage functions
  - Assertion enhancements
    - More assertion data types
    - More sampled value data types
    - Testing static class properties
    - Global clock redefined
    - Inferred clocks in sequences
    - Sequence method expressions
    - Final deferred immediate assertions
    - Fine-grained assertion control
  - Checker enhancements
    - Checker Output Arguments
    - More Checker Programming
  - VPI enhancements
    - 4+ extensions to support new features

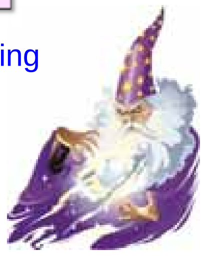
About the Author

- **Mr. Stuart Sutherland**
  - Has been using Verilog and SystemVerilog since 1988
  - Worked as a design engineer on military flight simulators
  - Applications Engineer for Gateway Design Automation, the founding company of Verilog
  - Editor of the IEEE SystemVerilog standard (and every previous version of the Verilog and SystemVerilog standards)
  - Author of books on Verilog, SystemVerilog and the Verilog PLI
  - Involved in the IEEE 1364 Verilog and IEEE 1800 SystemVerilog standardization
  - President of Sutherland HDL, Inc. – a company specializing in expert Verilog/SystemVerilog consulting and training services (founded in 1992)

Expert SystemVerilog Training

A copy of this presentation is available at www.sutherland-hdl.com/papers

- Sutherland HDL, Inc. provides SystemVerilog training
  - ✓SystemVerilog for Design and Synthesis
  - ✓SystemVerilog for Verification
  - ✓SystemVerilog Assertions
  - ✓SystemVerilog UVM
  - All training workshops are available on-site and as online *eTutored*™ training
  - Sutherland HDL instructors are expert trainers and experienced design and verification engineers



Sutherland HDL helps engineers become true SystemVerilog wizards!

visit www.sutherland-hdl.com for workshop descriptions