Free IEEE SystemVerilog-2012 LRM @
http://standards.ieee.org/getieee/1800/download/1800-2012.pdf

The New SystemVerilog 2012 Standard

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

World-class Verilog, SystemVerilog & OVM/UVM Training

Life is too short for bad or boring training!

Open Enrollment Verilog, SystemVerilog & UVM Training
Dates and info posted on the Sunburst Design web page

Acknowledgements

SystemVerilog-2012 Committee Feedback

Apologizes if I missed any committee member feedback

- Alex Gran Mentor
- Arturo Salz Synopsys
- Ben Cohen HDL Cohen
- Brad Pierce Synopsys
- Brandon Tipp Intel
- Chris Spear Synopsys
- Dave Rich Mentor
- Dennis Brophy Mentor
- Dimitry Korchemny Intel
- Ed Cerny Synopsys
- Erik Seligman Intel
- Gord Vreugdenhil Mentor
- John Havlicek Cadence
- Jonathan Bromley Verilab
- Kaiming Ho Fraunhofer
- Kevin Cameron Cameron EDA
- Rob Hughes Intel
- Samik Sengupta Synopsys
- Scott Little Intel
- Shalom Bresticker Intel
- Shekar Chetput Cadence
- Stu Sutherland Sutherland-HDL
- Surrendra Dudani Synopsys
- Tom Alsop Intel
- Tom Fitzpatrick Mentor

Additional SystemVerilog-2012 Resources

Sunburst Design

- IEEE SystemVerilog-2012 LRM
  standards.ieee.org/getieee/1800/download/1800-2012.pdf
  FREE PDF version - download it !!
- Stu Sutherland's Presentation: *A Summary of Changes in the Proposed SystemVerilog-2012 Standard*
  www.sutherland-hdl.com/papers/2012-DAC_What-is-new-in-SystemVerilog-2012.pdf
- Brad Pierce's Blog: *SV12 - what's new in SystemVerilog 2012?*
  bradpierce.wordpress.com/2013/03/02/sv12-whats-new-in-systemverilog-2012
- Dave Rich's Blog: *Get Ready for SystemVerilog-2012*
  go.mentor.com/ready-for-systemverilog-2012
- *List of 225 Changes in the SystemVerilog-2012 Standard*
  http://tinyurl.com/1800-2012-changelog

230 Modifications in SystemVerilog-2012

Sunburst Design

- 230 listed modifications in the SystemVerilog-2012 LRM
  www.eda.org/svdb/view_all_bug_page.php
- Breakdown of Modifications:
  - Enhancements - 35
  - Errata - 162
  - Clarifications - 33

Types of Modifications

SystemVerilog-2012

- 230 listed modifications in the SystemVerilog-2012 LRM
  www.eda.org/svdb/view_all_bug_page.php

- Breakdown of types of Modifications:
  - Assertion-related - 87 - 13 new enhancements
  - Design-related - 66 - 9 new enhancements
  - C-Language / DPI-related - 25 - 4 new enhancements
  - Discrete Real Modeling - 5 - 1 new enhancements
  - Verification-related - 46 - 8 new enhancements
  - Typos / Cleanup - 1 - 0 new enhancements

Discrete Real Modeling - 5 Modifications

SystemVerilog-2012

- Mantis Item of 1 new enhancements

major 3398 - User defined nets and resolution functions

- SV-DC Requirements:
  - Aggregate nets with real valued components, including constructs such as vectors, static arrays, structs, unions
  - Resolution functions for multiply driven aggregate nets
  - Unidirectional and bidirectional ports with aggregate nets
  - Atomic aggregate nets whose components are resolved jointly (correlated resolution)
  - Ability to represent X (unknown) and Z (undriven/high impedance) states for atomic aggregate nets

Discrete Real Modeling Work

SystemVerilog-2012

- The SV-DC work consists of two primary pieces:
  - User-defined nettypes and resolution functions (6.6.7)
  - Generic interconnect (6.6.8)

Nice summary comments from Scott Little of Intel

These features provide the users some nice capabilities, but it isn't a complete solution.

The SV-DC realized that they didn't have time to provide a complete solution but decided to provide some pieces to help out users willing to deal with the initial heavy handed restrictions.

The intention is to add the missing pieces (a type conversion mechanism) and explore lifting some of the restrictions (no partial assignment to user-defined nettypes) as real world use cases make the proper solution clear. Because the new functionality was being developed in committee, the desire was to be very conservative and relax the restrictions later when we have real world examples and implementations to better guide the work.

Why is the work valuable? It provides a much improved solution for those building real-valued models.

C / DPI - 25 Modifications

SystemVerilog-2012

- Mantis Items of 4 new enhancements

minor	3188	- Added VPI support to distinguish join, join_none, and join_any
minor	3459	- Remove redundant DPI section "H.6.6 Pure functions"
feature	3884	- Added VPI support for soft constraints
minor	4130	- Update VPI compatibilities & table for IEEE Std 1800-2012

All minor VPI support enhancements
except for soft-constraint support (new feature)

Assertion - 87 Modifications

SystemVerilog-2012

- Mantis Items of 13 new enhancements

feature	2093	- Checker construct should permit output arguments
feature	2206	- Checkers: Random sim of non-deterministic free variables
feature	2209	- Add optional event control to deferred assertions in 2005
major	2328	- Review and relax restrictions on data types in assertions
minor	2412	- Allow clock inference in sequences
feature	3202	- Clarifies $countbits, global clocking & other assertion functions
feature	3033	- Enhance checker modeling capabilities
minor	3037	- Introduce assertion system functions for 4-valued type support
minor	3069	- Multi-clock support - allow global clocking per module, interface, checker or program
minor	3191	- Allow sequence methods with sequence expressions
major	3206	- Deferred assertions are sensitive to glitches
feature	3295	- Need a way to control only asserts/covers/assume directives
minor	3474	- Allows assertion control over unique/priority violation reporting

Duplicate of Mantis 2476

SystemVerilog Assertions

- The SVA "circle of life"
  - Engineers get excited about SVA capabilities
  - Engineers take SVA training
  - Engineers start to use SVA
  - Engineers find SVA to be too verbose
  - Engineers abandon SVA

"The Disillusioned Design Engineer cycle!!"

aargh

How Much SVA Training Should Your Team Take?

- 2-day SystemVerilog Assertion (SVA) training is too much

Sunburst Design has been re-training SVA-trained engineers

- Inefficient SVA coding styles are shown in books and training
- Most engineers should take 2 hours of SVA training with labs

Learn the simple techniques

Reduce SVA coding efforts

Learn best-practice tricks

Learn styles that avoid mistakes

SVA Fundamentals

Guidelines

- Start learning and using SVA after 2 hours of training
- Use long, descriptive labels to:
  - document the assertions
  - accelerate debugging using waveform displays
- Use simple macros to:
  - efficiently add concise assertions
  - reduce assertion syntax errors
  - reduce assertion coding efforts
- Use concurrent assertions but avoid immediate assertions
- Use `|-> ##1` implications instead of `|=>` implications
- Use **`bind`** files to add assertions to a design

Design - 66 Modifications

SystemVerilog-2012

- Mantis Items of 9 new enhancements

feature	154	- Dual Data Rate (DDR) always_ff
feature	696	- Add parameterized tasks and functions
minor	931	- BNF should be hyperlinked
minor	1223	- Red hyperlinked BNF?
feature	1504	- Introduce parameterized structures
feature	2525	- Allow hierarchical references in $unit scope
minor	2734	- Mechanism to initialize an array to a constant value
minor	3750	- Update LRM & `begin_keywords with table of new keywords
minor	4126	- Allow for-loop initialization, step & termination stmnts to be null

Actually fixed by Mantis 2396 in SV2009

Parameterized Functions

- Additional descriptions:
  - **Brad Pierce** - *SV12 – deliver parameterized functions with let expressions*

    bradpierce.wordpress.com/2013/04/20/sv12-deliver-parameterized-functions-with-let-expressions

always_ff

For Dual Data Rate (DDR) Sequential Logic ??

- Possible future enhancement to synthesis tools ??

Currently illegal syntax for synthesis

No posedge (clk)
No negedge (clk)

```
module ddrff (
  output bit_t q,
  input  bit_t d, clk, rst_n);

  always_ff @(clk, negedge rst_n)
    if (!rst_n) q <= 0;
    else        q <= d;
endmodule
```

Remove posedge to permit triggering on both edges ??

always_ff shows designer's intent

```
always_ff @(edge clk, negedge rst_n)
```

Could this synthesize to a DDR flip-flop in an ASIC vendor library ??

Verification - 47 Modifications

SystemVerilog-2012

- Mantis Items of 8 new enhancements

feature 1356 - Multiple inheritance
major 2112 - Remove restrictions on NBA assignments to class members
feature 2506 - Non-trivial coverage space shapes and joint conditions are difficult to specify with covergroups
feature 2987 - Soft Constraints
feature 3001 - Proper Polymorphic behavior of instantiation
minor 3028 - Constraints for unique array elements
trivial 3298 - Use of 'this' in a coverpoint expression
major 4131 - function prototype syntax requires parentheses after function_identifier even if argument list is empty

SystemVerilog-2012

Miscellaneous Favorite Features

- As reported by SystemVerilog-2012 committee members

> My favorite enhancement in SV-2012 *is that it doesn't have AOP* ;-)
> - Tom Fitzpatrick - Mentor

This feeling was echoed in committee meetings by all of the developers !!

> *But my favorite thing out of that list is that only 35 are enhancements.* That is a testament to the stability users have demanded from their verification environment.
> - Dave Rich - Mentor

Also echoed by other developers on the committee

Unconstrained Randomization

Repetitive Random Values

```
class trans1;
  rand logic [1:0] a, b, c, d;


  function void post_randomize();
    $display("a=%h  b=%h  c=%h  d=%h",
             a, b, c, d);
  endfunction
endclass

module top;
  trans1 b1 = new();

  initial repeat(16) if(!b1.randomize())
          $error("Randomization Error");
endmodule
```

Four 2-bit variables

No randomization constraints

Rare, unique values!

Multiple, repeated random values

Rare, unique values!

```
a=0  b=2  c=0  d=2
a=0  b=1  c=0  d=0
a=1  b=0  c=1  d=3
a=0  b=2  c=0  d=3
a=2  b=0  c=3  d=1
a=1  b=0  c=0  d=1
a=3  b=3  c=1  d=1
a=2  b=0  c=2  d=1
a=3  b=0  c=1  d=3
a=0  b=2  c=1  d=3
a=1  b=0  c=1  d=2
a=0  b=1  c=1  d=2
a=0  b=1  c=2  d=1
a=0  b=3  c=2  d=3
a=1  b=0  c=1  d=3
a=2  b=1  c=0  d=1
```

Unique Constraint

Applied to a Randomization List

```
class trans2;
  rand logic [1:0] a, b, c, d;

  constraint c1 {unique{a,b,c,d};}

  function void post_randomize();
    $display("a=%h  b=%h  c=%h  d=%h",
             a, b, c, d);

  endfunction
endclass

module top;
  trans2 b1 = new();

  initial repeat(16) if(!b1.randomize())
          $error("Randomization Error");
endmodule
```

Four 2-bit variables

Create unique random values for each variable in the list

Each variable has a unique value

```
a=2  b=1  c=0  d=3
a=2  b=0  c=1  d=3
a=3  b=2  c=1  d=0
a=0  b=1  c=2  d=3
a=0  b=1  c=2  d=3
a=2  b=1  c=0  d=3
a=0  b=2  c=1  d=3
a=3  b=1  c=2  d=0
a=2  b=1  c=0  d=3
a=3  b=2  c=1  d=0
a=2  b=0  c=3  d=1
a=3  b=0  c=2  d=1
a=2  b=3  c=1  d=0
a=2  b=3  c=1  d=0
a=2  b=1  c=0  d=3
a=0  b=3  c=1  d=2
```


Unique Constraint

Applied to an Array

```
class trans3;
  rand logic [2:0] a [0:6];

  constraint c1 {unique{a};}

  function vo
    foreach(a

    $display(
  endfunction
endclass

module top;
  trans3 b1 =

  initial rep
          $er
endmodule
```

3-bit by 7-element rand-array declaration

Create unique random values in each array element

Each row has unique values

```
a[0]=0  a[1]=5  a[2]=7  a[3]=6  a[4]=3  a[5]=4  a[6]=1
a[0]=3  a[1]=7  a[2]=5  a[3]=6  a[4]=1  a[5]=2  a[6]=4
a[0]=3  a[1]=1  a[2]=7  a[3]=2  a[4]=6  a[5]=5  a[6]=4
a[0]=6  a[1]=5  a[2]=3  a[3]=2  a[4]=7  a[5]=4  a[6]=1
a[0]=7  a[1]=5  a[2]=3  a[3]=1  a[4]=6  a[5]=4  a[6]=0
a[0]=2  a[1]=3  a[2]=1  a[3]=4  a[4]=5  a[5]=7  a[6]=6
a[0]=5  a[1]=0  a[2]=2  a[3]=3  a[4]=6  a[5]=1  a[6]=4
a[0]=7  a[1]=4  a[2]=5  a[3]=2  a[4]=1  a[5]=6  a[6]=3
a[0]=1  a[1]=5  a[2]=4  a[3]=3  a[4]=6  a[5]=7  a[6]=0
a[0]=6  a[1]=3  a[2]=4  a[3]=7  a[4]=2  a[5]=1  a[6]=5
a[0]=4  a[1]=2  a[2]=7  a[3]=0  a[4]=3  a[5]=5  a[6]=6
a[0]=2  a[1]=0  a[2]=1  a[3]=6  a[4]=4  a[5]=5  a[6]=7
a[0]=2  a[1]=0  a[2]=4  a[3]=6  a[4]=7  a[5]=1  a[6]=5
a[0]=1  a[1]=4  a[2]=5  a[3]=6  a[4]=3  a[5]=2  a[6]=0
a[0]=1  a[1]=0  a[2]=4  a[3]=7  a[4]=3  a[5]=6  a[6]=2
a[0]=5  a[1]=7  a[2]=3  a[3]=2  a[4]=6  a[5]=0  a[6]=4
```

Unique Constraint

Applied to an Array - Randomization Error

```
class trans4;
  rand logic [2:0] a [0:8];

  constraint c1 {unique{a};}

  function void post_randomize();
    foreach(a[i]) $write("a[%1d]=%h  ",
                         i, a[i]);
    $display();
  endfunction
endclass

module top;
  trans4 b1 = new();

  initial repeat(16) if(!b1.randomize())

          $error("Randomization Error");
endmodule
```

3-bit by 9-element rand-array declaration

Create unique random values in each array element

"Randomization Error" *(Not enough unique values for each element of the array)*

Soft Constraints

Example with No Constraints

```
class trans1;
  rand  logic [2:0] a;
  randc logic [2:0] b;
        logic [2:0] c;



  function void post_randomize();
    $display("a=%h  b=%h  c=%h", a, b, c);
  endfunction
endclass

module top;
  trans1 b1 = new();

  initial repeat(16) if(!b1.randomize())

          $error("Randomization Error");
endmodule
```

3-bit rand variable

3-bit randc variable

3-bit non-rand variable

```
a=0  b=0  c=x
a=3  b=7  c=x
a=0  b=2  c=x
a=6  b=1  c=x
a=3  b=4  c=x
a=0  b=3  c=x
a=5  b=6  c=x
a=4  b=5  c=x
a=4  b=1  c=x
a=2  b=4  c=x
a=6  b=0  c=x
a=6  b=7  c=x
a=7  b=3  c=x
a=2  b=6  c=x
a=0  b=2  c=x
a=6  b=5  c=x
```

Soft Constraints

Example with Hard Constraints


```
class trans2;
  rand  logic [2:0] a;
  randc logic [2:0] b;
        logic [2:0] c;

  constraint c1 {a[0]=='0; b[0]=='1;}

  function void post_randomize();
    $display("a=%h  b=%h  c=%h", a, b, c);
  endfunction
endclass

module top;
  trans2 b1 = new();

  initial repeat(16) if(!b1.randomize())

          $error("Randomization Error");
endmodule
```

3-bit rand variable

3-bit randc variable

3-bit non-rand variable

```
a=4  b=5  c=x
a=2  b=7  c=x
a=4  b=1  c=x
a=2  b=3  c=x
a=4  b=7  c=x
a=0  b=1  c=x
a=0  b=5  c=x
a=2  b=3  c=x
a=0  b=3  c=x
a=4  b=5  c=x
a=4  b=1  c=x
a=0  b=7  c=x
a=2  b=3  c=x
a=4  b=5  c=x
a=4  b=1  c=x
a=4  b=7  c=x
```


Soft Constraints

Example with Conflicting Hard Constraints


```
class trans3;
  rand  logic [2:0] a;
  randc logic [2:0] b;
        logic [2:0] c;

  constraint c1 {a[0]=='0; b[0]=='1;}

  function void post_randomiz
    $display("a=%h  b=%h  c=%
  endfunction
endclass

module top;
  trans3 b1 = new();

  initial repeat(16) if(!(b1.randomize()
                     with {a[0]=='1;}))
          $error("Randomization Error");
endmodule
```

3-bit rand variable

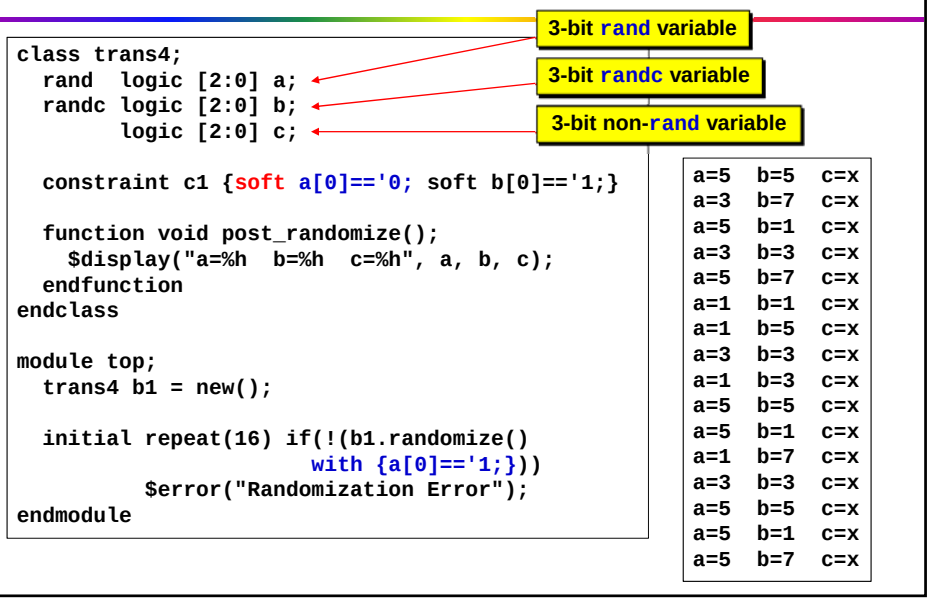
3-bit randc variable

3-bit non-rand variable

```
** Error: Randomization Error
   Scope: top File: trans3.sv Line: 18
** Error: Randomization Error
   Scope: top File: trans3.sv Line: 18
**
** (repeated 16 times)
```

Soft Constraints

Example with Soft & Hard Constraints

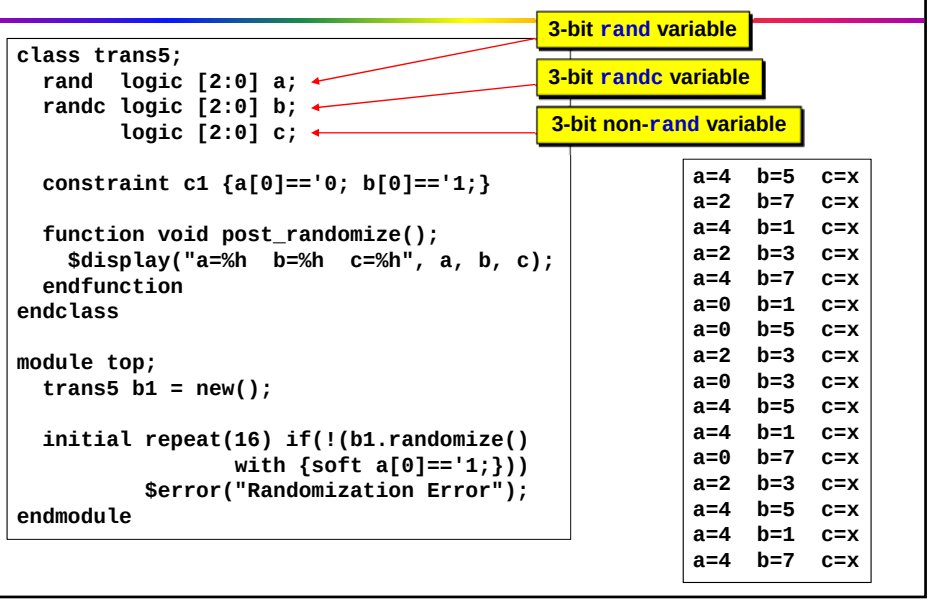



```
class trans4;
  rand  logic [2:0] a;
  randc logic [2:0] b;
        logic [2:0] c;

  constraint c1 {soft a[0]=='0; soft b[0]=='1;}

  function void post_randomize();
    $display("a=%h  b=%h  c=%h", a, b, c);
  endfunction
endclass

module top;
  trans4 b1 = new();

  initial repeat(16) if(!(b1.randomize()
                          with {a[0]=='1;}))
          $error("Randomization Error");
endmodule
```

```
a=5  b=5  c=x
a=3  b=7  c=x
a=5  b=1  c=x
a=3  b=3  c=x
a=5  b=7  c=x
a=1  b=1  c=x
a=1  b=5  c=x
a=3  b=3  c=x
a=1  b=3  c=x
a=5  b=5  c=x
a=5  b=1  c=x
a=1  b=7  c=x
a=3  b=3  c=x
a=5  b=5  c=x
a=5  b=1  c=x
a=5  b=7  c=x
```


Soft Constraints

Example #2 with Soft & Hard Constraints

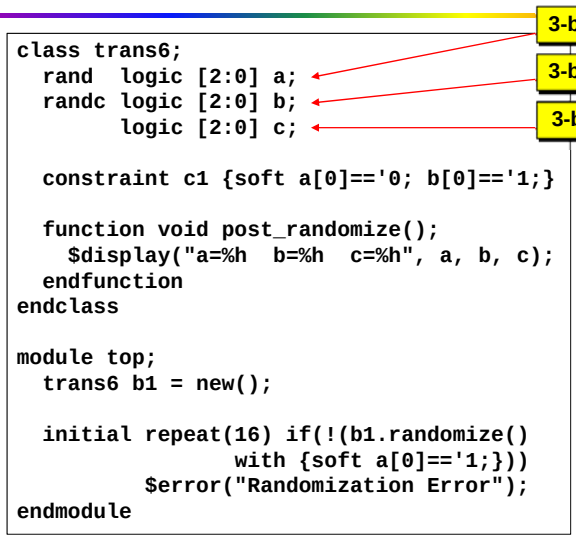



```
class trans5;
  rand  logic [2:0] a;
  randc logic [2:0] b;
        logic [2:0] c;

  constraint c1 {a[0]=='0; b[0]=='1;}

  function void post_randomize();
    $display("a=%h  b=%h  c=%h", a, b, c);
  endfunction
endclass

module top;
  trans5 b1 = new();

  initial repeat(16) if(!(b1.randomize()
                  with {soft a[0]=='1;}))
          $error("Randomization Error");
endmodule
```

```
a=4  b=5  c=x
a=2  b=7  c=x
a=4  b=1  c=x
a=2  b=3  c=x
a=4  b=7  c=x
a=0  b=1  c=x
a=0  b=5  c=x
a=2  b=3  c=x
a=0  b=3  c=x
a=4  b=5  c=x
a=4  b=1  c=x
a=0  b=7  c=x
a=2  b=3  c=x
a=4  b=5  c=x
a=4  b=1  c=x
a=4  b=7  c=x
```

Soft Constraints

Example with Prioritized Soft Constraints




```
class trans6;
  rand  logic [2:0] a;
  randc logic [2:0] b;
        logic [2:0] c;

  constraint c1 {soft a[0]=='0; b[0]=='1;}

  function void post_randomize();
    $display("a=%h  b=%h  c=%h", a, b, c);
  endfunction
endclass

module top;
  trans6 b1 = new();

  initial repeat(16) if(!(b1.randomize()
                with {soft a[0]=='1;}))
          $error("Randomization Error");
endmodule
```

```
a=5  b=5  c=x
a=3  b=7  c=x
a=5  b=1  c=x
a=3  b=3  c=x
a=5  b=7  c=x
a=1  b=1  c=x
a=1  b=5  c=x
a=3  b=3  c=x
a=1  b=3  c=x
a=5  b=5  c=x
a=5  b=1  c=x
a=1  b=7  c=x
a=3  b=3  c=x
a=5  b=5  c=x
a=5  b=1  c=x
a=5  b=7  c=x
```


Multiple Inheritance / Interface Classes

- Additional descriptions:
  - **Dave Rich** - *The Problems with Lack of Multiple Inheritance in SystemVerilog and a Solution*

    bradpierce.files.wordpress.com/2010/03/multiple_inheritance_sv.pdf

Interface Class Rules

KEY: In the text below:

Text	Actual Keyword
`extend`	`extends`
`implement`	`implements`

- An **`interface`** class
  - may declare **`pure virtual`** prototypes
  - may **`extend`** zero or more **`interface`** classes
  - may extend multiple other **`interface`** classes
  - may not **`implement`** an **`interface`** class
  - may not **`extend`** a class or **`virtual`** class
  - may not **`implement`** a class or **`virtual`** class
  - may not implement any class menthods

Interface Classes & Multiple Inheritance

Example: ifc1_pass.sv

```
interface class ifbase;
  pure virtual function void print();
endclass

class base implements ifbase;
  virtual function void print();
    $display("%m: print() method");
  endfunction
endclass

module top;
  base b1 = new();

  initial begin
    b1.print();
  end
endmodule
```

`Interface` classes may declare **`pure virtual`** prototypes

Provide the implementation

top module and simulation output

```
ifc1_pass_sv_unit.base.print: print() method
```

Interface Classes & Multiple Inheritance

Example: ifc2_pass.sv

`interface` classes may `extend` zero or more `interface` classes

```
interface class ifbase;
  pure virtual function void print();
endclass

interface class GetImp#(type GET_T = int)
                        extends ifbase;
  pure virtual function GET_T get();
endclass


class base implements GetImp;
  virtual function void print();
    $display("%m: print() method");
  endfunction

  virtual function int get();
    $display("%m: get() method");
  endfunction
endclass
```

Provide the implementation

```
module top;
  base b1 = new();

  initial begin
    b1.print();
    void'(b1.get());
  end
endmodule
```

```
ifc2_pass_sv_unit.base.print: print() method
ifc2_pass_sv_unit.base.get: get() method
```


Interface Classes & Multiple Inheritance

Example: ifc3_pass.sv

```
interface class ifbase1;
  pure virtual function void print();
endclass

interface class ifbase2;
  pure virtual function void showit();
endclass

interface class GetImp#(type GET_T = int)
                  extends ifbase1, ifbase2;
  pure virtual function GET_T get();
endclass
```

`Interface` classes may `extend` multiple other `interface` classes

```
class base implements GetImp;
  virtual function void print();
    $display("%m: print() method");
  endfunction

  virtual function void showit();
    $display("%m: showit() method");
  endfunction

  virtual function int get();
    $display("%m: get() method");
  endfunction
endclass
```

Interface Classes & Multiple Inheritance

Example: ifc3_pass.sv

```
class base implements GetImp;
  virtual function void print();
    $display("%m: print() method");
  endfunction

  virtual function void showit();
    $display("%m: showit() method");
  endfunction

  virtual function int get();
    $display("%m: get() method");
  endfunction
endclass
```

```
module top;
  base b1 = new();

  initial begin
    b1.print();
    b1.showit();
    void'(b1.get());
  end
endmodule
```

The example ran !!

```
ifc3_pass_sv_unit.base.print: print() method
ifc3_pass_sv_unit.base.showit: showit() method
ifc3_pass_sv_unit.base.get: get() method
```


Interface Classes & Multiple Inheritance

Example: ifc4_error.sv

`Interface` classes may not `implement` other `interface` classes

```
interface class ifbase;
  pure virtual function void print();
endclass

interface class GetImp#(type GET_T = logic) implements ifbase;
  pure virtual function GET_T get();
endclass
```

```
Error: ifc4_error.sv(5): Interface classes may
     not have an implements specification.
```

Interface Classes & Multiple Inheritance

Example: ifc5_error.sv

```
virtual class vbase;
  virtual function void print();
    $display("Virtual base class");
  endfunction
endclass

interface class GetImp#(type GET_T = logic) extends vbase;
  pure virtual function GET_T get();
endclass
```

Interface classes may not extend a class or a virtual class

```
Error: ifc5_error.sv(7): Illegal reference to non-interface class
            vbase in interface class extends clause.
```


Interface Classes & Multiple Inheritance

Example: ifc6_error.sv

```
virtual class vbase;
  virtual function void print();
    $display("Virtual base class");
  endfunction
endclass

interface class GetImp#(type GET_T = logic) implements vbase;
  pure virtual function GET_T get();
endclass
```

Interface classes may not implement a class or a virtual class

```
Error: ifc6_error.sv(7): Interface classes may not have
       an implements specification.
Error: ifc6_error.sv(7): Illegal reference to non-interface
       class vbase in implements clause.
```

Interface Classes & Multiple Inheritance

Example: ifc7_error.sv

```
interface class GetImp#(type GET_T = logic);
  virtual function GET_T get();
  endfunction
endclass
```

Interface classes may not implement any methods

```
Error: Method get of interface class
    GetImp must be pure virtual.
```


Interface Class Rules

KEY: In the text below:

Text	Actual Keyword
extend	extends
implement	implements

- A class or **virtual** class
  - may not **extend** an **interface** class
  - may **implement** zero or more **interface** classes
  - may **extend** at most one other class or **virtual** class
  - may not **implement** a class or **virtual** class
  - may simultaneously **extend** a class and **implement** interface classes

Sunburst Design

TLM FIFO with Multiple Inheritance

Interface Classes & Multiple Inheritance

Example: ifclass_TLMfifo.sv

Sunburst Design

```
interface class PutImp#(type PUT_T = int);
  pure virtual task put(PUT_T a);
endclass

interface class GetImp#(type GET_T = int);
  pure virtual task get(output GET_T a);
endclass

class MyFIFO #(type T = int, int DEPTH=0);
  mailbox #(T) queue;
endclass
```

PutImp#() interface class

GetImp#() interface class

MyFIFO class

Interface Classes & Multiple Inheritance

Example: ifclass_TLMfifo.sv

```
class fifo #(type T = int, int DEPTH=0)
  extends MyFIFO#(T)
  implements PutImp#(int), GetImp#(int);

  function new();
    queue = new(DEPTH);
  endfunction

  virtual task put(int a);
    queue.put(a);
  endtask

  virtual task get(output int a);
    queue.get(a);
  endtask
endclass
```

Inheritance of `mailbox`

```
mailbox #(T) queue;
```

Implementation for `put()` method

```
pure virtual task put(PUT_T a);
```

Implementation for `get()` method

```
pure virtual task get(output GET_T a);
```


Interface Classes & Multiple Inheritance

Example: ifclass_TLMfifo.sv

```
module top;
  fifo f = new();
  int  i, j;

  initial begin
    $timeformat(-9,0,"ns", 4);
    fork
      repeat(5) begin
        #1;
        i = $urandom_range(0,99);
        $display("%t: FIFO: put %2d", $time, i);
        f.put(i);
      end
      repeat(5) begin
        #2;
        f.get(j);
        $display("%t: FIFO:         get %2d", $time, j);
      end
    join
    $finish;
  end
endmodule
```

```
 1ns: FIFO: put 34
 2ns: FIFO:         get 34
 2ns: FIFO: put  7
 3ns: FIFO: put 87
 4ns: FIFO:         get  7
 4ns: FIFO: put 43
 5ns: FIFO: put 14
 6ns: FIFO:         get 87
 8ns: FIFO:         get 43
10ns: FIFO:         get 14
```

Nothing More To Do! (???)

Sunburst Design

- Patent Office Quote

> ***"Everything that can be invented has been invented."***
>
> ***Charles H. Duell, Commissioner, U.S. patent office, 1899 (attributed)***

NOTE: Debunked - This quotation is not really the words of Mr. Duell.
(www.quotationspage.com/quote/22779.html)

Point is, nobody should ever claim that all important work has been done

More Features Proposed

Sunburst Design

- **`alwaysx`** - X-trapping ← **Don't lose the X's**
- Connectivity checking ← **Check to make sure each signal has:**
  - ***Driving source***
  - ***Destination receiver***

```
module ...
  input () scanin;
  ...
```

- `default_nettype / `default_type logic ← **Cliff's request**
  - Implicit variables for the LHS of procedural assignments ← **Stu's request**
- Signed operators -vs- sign data types
  - **HDL**
  - **Software DL**

```
<+>  <->
<*>  <*,ieee754>
<etc.>
```

- Default **`ignore_bins`** in cross coverage ← **Stu's request**

Free IEEE SystemVerilog-2012 LRM @
http://standards.ieee.org/getieee/1800/download/1800-2012.pdf

The New SystemVerilog 2012 Standard

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

World-class Verilog, SystemVerilog & OVM/UVM Training

Life is too short for bad or boring training!

Open Enrollment Verilog, SystemVerilog & UVM Training
Dates and info posted on the Sunburst Design web page