

SystemVerilog, ModelSim, and You

Is there anything in SystemVerilog
useful in your work?

Stuart Sutherland
Sutherland HDL, Inc.



Agenda

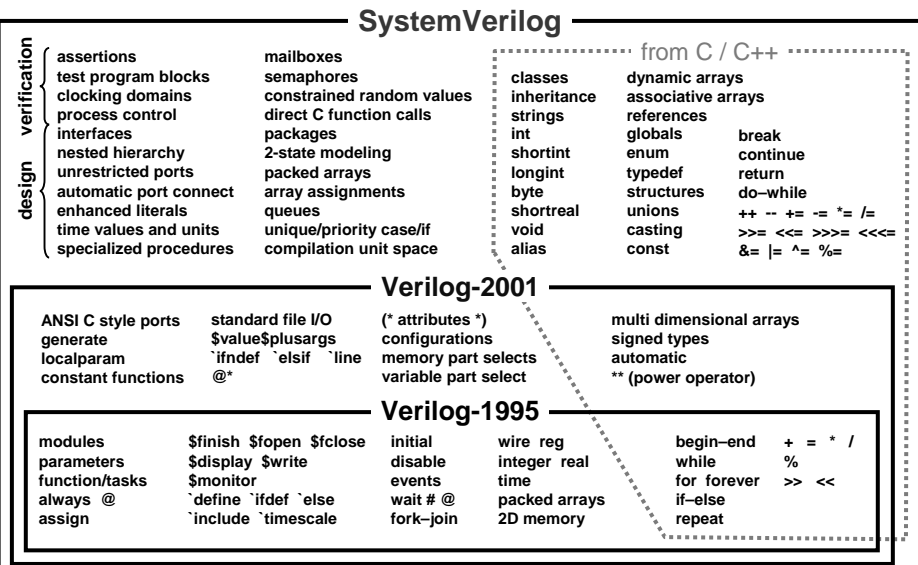
- ❑ Overview of SystemVerilog
- ❑ Convenience enhancements to Verilog
- ❑ RTL modeling enhancements to Verilog
- ❑ Abstract modeling enhancements to Verilog
- ❑ Verification enhancements to Verilog
- ❑ ModelSim support for SystemVerilog
- ❑ Suggestions on adopting SystemVerilog
- ❑ Conclusions

What is SystemVerilog?

- **SystemVerilog extends of the IEEE 1364 Verilog standard**
 - New design modeling capabilities
 - Abstract C language data types
 - More accurate RTL coding
 - Interfaces for communication
 - New verification capabilities
 - Assertions
 - Race-free test benches
 - Object-oriented test programs
- **SystemVerilog is the next generation of the Verilog standard**
 - Gives Verilog a much higher level of modeling abstraction
 - Gives Verilog advanced capabilities for design verification

3
SS, SystemVerilog, ModelSim, and You, April 2004

Mile High View of SystemVerilog



4
SS, SystemVerilog, ModelSim, and You, April 2004

SystemVerilog is an Evolution

- SystemVerilog evolves Verilog, rather than replacing it
 - Gives engineers the best of Verilog and C and Vera

Standard Verilog HDL

- Familiar
- Concurrency
- Proven to work

C language features

- Structures
- Globals
- ++ operator
- User-defined types
- and much more...

```
int clock; //global variables
module my_system (...);
  always @(posedge clock)
    case(instruction)
      ...
      ROR: out = rotate(...);
    endcase

  enum {ADD, SUB, ROR} instruction;
  struct {int word1, word2;} packet;

  function int rotate (int data_in, n);
    int temp;
    for (int i=0; i<n; i++)
      ...
    return(temp);
  endfunction
endmodule
```

*This is easy,
it's just like
using Verilog,
only more!*



5
SS, SystemVerilog, ModelSim, and You, April 2004

What's Next

- ✓ Objectives and caveats
- ❑ Convenience enhancements to Verilog
- ❑ RTL modeling enhancements to Verilog
- ❑ Abstract modeling enhancements to Verilog
- ❑ Verification enhancements to Verilog
- ❑ ModelSim support for SystemVerilog
- ❑ Suggestions on adopting SystemVerilog
- ❑ Conclusions

6
SS, SystemVerilog, ModelSim, and You, April 2004

Enhanced Specification of Time Units and Time Precision

- In Verilog, time units are a module property
 - Declared with the ``timescale` compiler directive

```
forever #5 clock = ~clock; 5 what?
```

- SystemVerilog adds:
 - Time units can be specified as part of the time value

```
forever #5ns clock = ~clock;
```

- Module time units and precision can be specified with keywords

```
module my_chip (...);  
  timeunit 1ns;  
  timeprecision 10ps;  
  ...  
endmodule
```

7
SS, SystemVerilog, ModelSim, and You, April 2004

Enhanced Literal Value Assignments

- In Verilog, the vector size must be hard coded in order to fill all bits with 1

```
reg [127:0] data_bus;  
data_bus = 128'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF;
```

- SystemVerilog enhances assignments of a literal value
 - All bits of a vector can be filled with a literal 1-bit value
 - '0 fills all bits on the left-hand side with 0
 - '1 fills all bits on the left-hand side with 1
 - 'z fills all bits on the left-hand side with z
 - 'x fills all bits on the left-hand side with x

```
reg [127:0] data_bus;  
data_bus = '1; //set all bits of data_bus to 1
```

8
SS, SystemVerilog, ModelSim, and You, April 2004

The SystemVerilog logic Data Type

- In Verilog, the term “reg” confuses new users

```
reg [31:0] sum;  
always @(a or b)  
    sum = a + b;
```

- The name would seem to infer a hardware *register*
- In reality, `reg` is a general purpose variable that can represent either combinational logic or sequential logic



- SystemVerilog’s 4-state `logic` type is identical to a `reg`

- `logic` is a more intuitive name for new Verilog users
 - Verilog has other synonym keywords, such as `wire` and `tri`

```
logic [31:0] sum;  
always @(a or b)  
    sum = a + b;
```

9
SS, SystemVerilog, ModelSim, and You, April 2004

SystemVerilog Relaxes Verilog Variable Semantic Rules

- Verilog has strict rules on when to use a variable (eg. `reg`) and when to use a net (e.g. `wire`)

- Context dependent
- A variable cannot be “driven” by a continuous assignment or an output port



- SystemVerilog allows variables to be used in the same places a net can be used

- Limited to a single driver type (procedural, continuous or output of a module/primitive instance)
- The 1-driver limit prevents inadvertent shared variable behavior where wired-logic resolution is needed

10
SS, SystemVerilog, ModelSim, and You, April 2004

An Example of Using Variables With SystemVerilog's Relaxed Rules

- The same data type can be used for the entire model
 - With the restriction that all inputs have a single driver

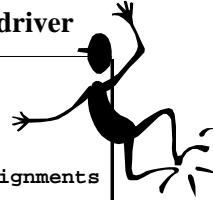
```
module compare (output logic lt, eq, gt,
               input logic [63:0] a, b);

  always @(a or b)
    if (a < b) lt = 1'b1; //procedural assignments
    else      lt = 1'b0;

  assign gt = (a > b); //continuous assignments

  comparator u1 (eq, a, b); //module instance

endmodule
```



The restriction of a single driver can prevent unintentional design errors!

11
SS, SystemVerilog, ModelSim, and You, April 2004

Relaxed Rules for Passing Values Through Module Ports

- Verilog restricts the data types that can be connected to module ports
 - Only net types on the receiving side
 - Nets, regs or integers on the driving side
 - Choosing the correct type frustrates Verilog modelers
- SystemVerilog removes all restrictions on port connections
 - Any data type is allowed on either side of the port
 - Read numbers can be passed through ports
 - Arrays can be passed through ports
 - Structures can be passed through ports



12
SS, SystemVerilog, ModelSim, and You, April 2004

Module Instance Port Connection Shortcuts

- Verilog module instances can use port-name connections
 - Must name both the port and the net connected to it

```

module dff (output q, qb,
           input clk, d, rst, pre);
...
endmodule
module chip (output [3:0] q,
            input [3:0] d, input clk, rst, pre);
  dff dff1 (.clk(clk), .rst(rst), .pre(pre), .d(d[0]), .q(q[0]));

```

can be verbose and redundant

- SystemVerilog adds `.name` and `.*` shortcuts

- `.name` connects a port to a net of the same name

```

dff dff1 (.clk, .rst, .pre, .d(d[0]), .q(q[0]));

```

- `.*` automatically connects all ports and nets that have the same name

```

dff dff1 (.*, .q(q[0]), .d(d[0]));

```

13
SS, SystemVerilog, ModelSim, and You, April 2004

Task/Function Arguments: Passing By Name

- In Verilog:

- Values are passed to tasks and functions by position

How can I know if `stack` and `data_bus` are in the right order?

```

always @(posedge clock)
  result = subtractor( stack, data_bus );

function integer subtractor(input integer a, b);
  subtractor = a - b;
endfunction

```



- In SystemVerilog

- Values can be passed using the formal argument name

```

always @(posedge clock)
  result = subtractor( .b(stack), .a(data_bus) );

function int subtractor(int a, b);
  return(a - b);
endfunction

```

Uses same syntax as named module port connections

`.name` and `.*` connections can also be used

14
SS, SystemVerilog, ModelSim, and You, April 2004

Block Names and Statement Labels

- Verilog allows a statement group to have a name
 - Identifies all statements within the block
 - Creates a new level of model hierarchy

```
begin: block1 ... end
```

- SystemVerilog adds:
 - A name can be specified after the end keyword
 - Documents which statement group is being ended

```
begin: block2 ... end: block2
```

- Specific statements can be given a “label”
 - Identifies a single statement
 - Can aid in debugging, coverage reporting, etc.

```
shifter: for (i=15; i>0; i--)
```

15
SS, SystemVerilog, ModelSim, and You, April 2004

Named End Statements

- SystemVerilog also extends to ability to specify an ending name with `endmodule`, `endinterface`, `endprimitive`, `endprogram`, `endtask`, `endfunction`, `endclass`, `endproperty`, and `endsequence`
 - The ending name must match the name used with the corresponding beginning of the code block

```
module my_chip (...);  
  ...  
  task get_data (...)  
    ...  
  endtask: get_data  
endmodule: my_chip
```

Specifying ending names helps to make large blocks of code more readable, but does not affect functionality in any way

16
SS, SystemVerilog, ModelSim, and You, April 2004

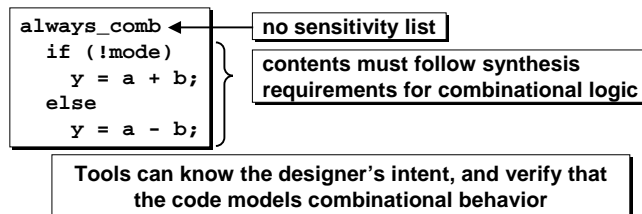
What's Next

- ✓ Objectives and caveats
- ✓ Convenience enhancements to Verilog
- ❑ **RTL modeling enhancements to Verilog**
- ❑ Abstract modeling enhancements to Verilog
- ❑ Verification enhancements to Verilog
- ❑ ModelSim support for SystemVerilog
- ❑ Suggestions on adopting SystemVerilog
- ❑ Conclusions

17
SS, SystemVerilog, ModelSim, and You, April 2004

Hardware Specific Procedural Blocks

- **The Verilog `always` procedural block is general purpose block**
 - Used to model combinational, latched, and sequential logic
 - Software tools must “infer” (*guess*) what type of hardware an engineer intended based on procedure content and context
- **SystemVerilog adds special hardware-oriented procedures: `always_ff`, `always_comb`, and `always_latch`**
 - Simulation, synthesis and formal tools to use same rules
 - Tools can check that designer’s intent has been modeled



18
SS, SystemVerilog, ModelSim, and You, April 2004

Unique and Priority Decisions

- Verilog defines that `if...else...if` decisions and `case` statements execute with priority encoding...
 - In simulation, only the first matching branch is executed
 - Synthesis will infer parallel execution based on context
 - Parallel evaluation after synthesis *may* causes a mismatch in pre-synthesis and post-synthesis simulation results
- SystemVerilog adds unique and priority decision modifiers
 - Priority-encoded or parallel evaluation can be explicitly defined for both simulation and synthesis
 - Software tools can warn if `case` or `if...else` decisions do not match the behavior specified

```
unique case (state)
  WAIT: ...
  LOAD: ...
  READY: ...
endcase
```

- Can be evaluated in parallel
- Will get an error if no branch is true

19
SS, SystemVerilog, ModelSim, and You, April 2004

New Operators

- Verilog does not have increment and decrement operators

```
for (i = 0; i <= 255; i = i + 1)
  ...
```

- SystemVerilog adds:

- ++ and -- increment and decrement operators
- +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, <<<=, >>>= assignment operators

```
for (i = 0; i <= 255; i++)
  ...
```

Hooray!



20
SS, SystemVerilog, ModelSim, and You, April 2004

The 2-state bit Data Type

- Verilog uses 4-state logic
 - Can be overhead to simulation performance
 - Most hardware can be modeled with 2-state logic
- Some simulator's "fake" 2-state with compilers
 - Not portable to other tools (not a standard)
 - Can have side affects if part of the design (or test bench) requires 3-state or 4-state logic
- The SystemVerilog 2-state bit type
 - Allows mixing 2-state and 4-state in the same design
 - Will work the same on all SystemVerilog simulators

```
bit [31:0] sum;  
always @(a or b)  
    sum = a + b;
```

21
SS, SystemVerilog, ModelSim, and You, April 2004

Enumerated Types

- With Verilog, constants must be used to give names to values

```
reg [2:0] traffic_light;  
parameter red    = 0;  
parameter green  = 1;  
parameter yellow = 2;  
  
always @(posedge clock)  
    if (traffic_light == red)  
        ...
```

The variable traffic_light could be assigned values other than red, green or yellow

- SystemVerilog adds enumerated types, using enum, as in C

```
enum {red, green, yellow} traffic_light;  
always @(posedge clock)  
    if (traffic_light == red)  
        ...
```

Simplifies declaration of named values

Limits the legal values of a variable

22
SS, SystemVerilog, ModelSim, and You, April 2004

User-defined Types

- Verilog does not have user-defined data types
- SystemVerilog adds user-defined types
 - Uses the `typedef` keyword, as in C

```
typedef enum {FALSE, TRUE} boolean;  
boolean ready; //variable "ready" can be FALSE or TRUE
```

```
enum {WAIT, LOAD, READY} states_t;  
states_t state, next_state;
```

23
SS, SystemVerilog, ModelSim, and You, April 2004

For Loop Variables

- In Verilog, the variable used to control a `for` loop must be declared prior to the loop

```
integer i;  
initial begin  
    for (i=0; i<= 255; i++)  
        ...  
end
```

i must be declared outside the loop

- SystemVerilog allows the declaration of the `for` loop variable within the `for` loop itself

```
initial  
begin  
    for (int i=0; i<= 255; i++)  
        ...  
end
```

i can be declared within the loop

- Makes the loop variable local to the loop
- Prevents conflicts between multiple `for` loops

24
SS, SystemVerilog, ModelSim, and You, April 2004

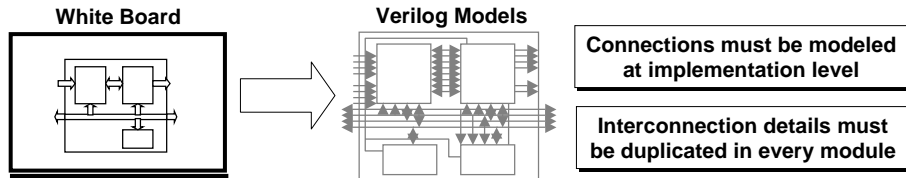
What's Next

- ✓ Objectives and caveats
- ✓ Convenience enhancements to Verilog
- ✓ RTL modeling enhancements to Verilog
- ❑ **Abstract modeling enhancements to Verilog**
- ❑ Verification enhancements to Verilog
- ❑ ModelSim support for SystemVerilog
- ❑ Suggestions on adopting SystemVerilog
- ❑ Conclusions

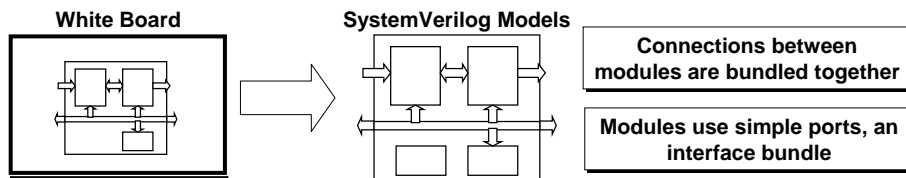
25
SS, SystemVerilog, ModelSim, and You, April 2004

Interfaces

■ Verilog connects models using detailed module ports



■ SystemVerilog adds interfaces



26
SS, SystemVerilog, ModelSim, and You, April 2004

Interfaces Simplify Module Interconnections

```
interface chip_bus (input bit clk);
    bit request, grant, ready;
    bit [47:0] address;
    bit [63:0] data;
endinterface
```

Connection details are in the interface

```
module CPU (chip_bus io);
    ...
endmodule
```

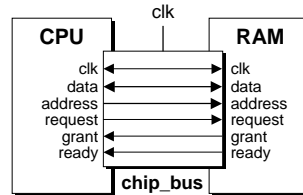
Modules do not duplicate connection detail

```
module RAM(chip_bus pins);
    ...
endmodule
```

```
module top;
    bit clk = 0;
    chip_bus a(clk); //instantiate the interface

    RAM mem(a); //connect interface to module instance
    CPU cpu(a); //connect interface to module instance
endmodule
```

Netlists do not duplicate connection detail



27
SS, SystemVerilog, ModelSim, and You, April 2004

Abstract Data Types

- Verilog has hardware-centric data types
 - Intended to represent real connections in a chip or system
 - At the system and RTL level, models only need 2-state logic
 - Tri-state busses are the only place 4-state logic is needed
- SystemVerilog adds several new data types
 - C-like data types create a bridge between C and Verilog
 - byte — an 8-bit 2-state integer
 - shortint — a 16-bit 2-state integer, similar to a C short
 - int — a 32-bit 2-state integer, similar to a C int
 - longint — a 64-bit 2-state integer, similar to a C longlong
 - shortreal — a 32-bit single-precision floating point, a C float
 - void — no value (used for function returns)

28
SS, SystemVerilog, ModelSim, and You, April 2004

Structures

■ SystemVerilog adds C-like structures to Verilog

A structure is a collection of variables that can be different types and sizes

```
struct {  
    real r0, r1;  
    int i0, i1;  
    bit [15:0] opcode;  
} instruction_word;  
...  
instruction_word.opcode = 16'hF01E;
```

The structure declaration is the same as in C

- Can be used to bundle several variables into one object
- Can assign to individual signals within the structure
- Can assign to the structure as a whole
- Can pass structures through ports and to tasks or functions

29
SS, SystemVerilog, ModelSim, and You, April 2004

Unions

■ A union is a single element that allows the storage of different data types in the same space

- Can be used to define storage before the type of the value to be stored is known
- Can be used to define storage where the type of the value stored may change during run time

```
union {  
    int i;  
    real r;  
} data;  
...  
data.i = 5;  
$display("data is %d", data.i);  
data.r = 2.01;  
$display("now data is %f", data.r);
```

"data" can only store one value, but the type of the value can be int or real

30
SS, SystemVerilog, ModelSim, and You, April 2004

Unpacked Arrays

- An “unpacked array” is an array of nets or variables
 - The signal can be any data type
 - The array can have any number of dimensions

```
wire n [0:1023];
```

a 1-dimensional “unpacked array” of 1024 1-bit nets

```
real r [0:1023];
```

a 1-dimensional “unpacked array” of 1024 real variables

```
int a [0:7][0:7][0:7];
```

a 3-dimensional “unpacked array” of 32-bit int variables

Unpacked array dimensions come after the array name (as in Verilog)

- In Verilog:
 - Only one element within an array can be accessed at a time
- SystemVerilog adds:
 - The entire array can be referenced (`my_array = your_array`)
 - Slices of arrays can be accessed

31
SS, SystemVerilog, ModelSim, and You, April 2004

Initializing and Assigning to Unpacked Arrays

- Unpacked arrays can be assigned using a list of values in { } braces for each array dimension (similar to C)

```
int d [0:1][0:3] = { {7,3,0,5}, {2,0,1,6} };
```

The { } braces are the C array initialize tokens,
not the Verilog concatenate operator!

Must have a set of braces for each array dimension

d[0][0] is initialized to 7
d[0][1] is initialized to 3
d[0][2] is initialized to 0
d[0][3] is initialized to 5
d[1][0] is initialized to 2
d[1][1] is initialized to 0
d[1][2] is initialized to 1
d[1][3] is initialized to 6

- A replicate operator can be used to full unpacked arrays

```
int d [0:1][0:3] = { 2{7,3,0,5} };
```

- A default assignment can also be used

```
int d [0:1][0:3] = { default:'1 };
```

d[0][0] is initialized to 7
d[0][1] is initialized to 3
d[0][2] is initialized to 0
d[0][3] is initialized to 5
d[1][0] is initialized to 7
d[1][1] is initialized to 3
d[1][2] is initialized to 0
d[1][3] is initialized to 5

32
SS, SystemVerilog, ModelSim, and You, April 2004

Jump Statements

- Verilog uses the `disable` statement as a go-to
 - Causes a named group of statements to jump to the end
- SystemVerilog adds C language jump statements to Verilog
 - `break` — works like the C break
 - `continue` — works like the C continue
 - `return(<value>)` — return from a non-void function
 - `return` — return from a task or void function

```
task send_packet(...);
  if (^data == 1'bx) begin
    $display("Error...");
    return; //abort task
  end
  ...
endtask
```

```
function real absolute(input real a);
  if (a >= 0.0) return(a);
  else      return(-a);
endfunction
```

The return statement allows terminating a task or function before reaching the end

33
SS, SystemVerilog, ModelSim, and You, April 2004

Task/Function Arguments: Passing By Reference

- In Verilog:
 - Inputs are copied into tasks and functions
 - Outputs are copied out of tasks

```
always @(posedge clock)
  result = subtractor( data_bus, stack );
function integer subtractor(input integer a, b);
  ...
```

- In SystemVerilog:
 - Task/function arguments can “reference” to calling argument
 - Uses the keyword `ref` instead of `input`, `output` or `inout`

```
always @(posedge clock)
  result = subtractor( data_bus, stack );
function int subtractor(int a, ref b);
  ...
```

The function receives a pointer to “stack” in the calling scope (note: the C “&” is not used)

34
SS, SystemVerilog, ModelSim, and You, April 2004

What's Next

- ✓ Objectives and caveats
- ✓ Convenience enhancements to Verilog
- ✓ RTL modeling enhancements to Verilog
- ✓ Abstract modeling enhancements to Verilog
- ❑ **Verification enhancements to Verilog**
- ❑ ModelSim support for SystemVerilog
- ❑ Suggestions on adopting SystemVerilog
- ❑ Conclusions

35
SS, SystemVerilog, ModelSim, and You, April 2004

SystemVerilog Assertions

■ SystemVerilog adds assertion syntax and semantics

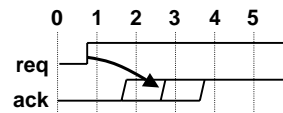
- Immediate assertions test for a condition at the current time

```
always @(state)
  assert (reset && (state != RST)) else $fatal);
```

generate a fatal error
if reset is true
and not in the reset state

- Concurrent assertions test for a sequence of events spread over time

```
sequence req_ack;
  @(posedge clk) req ##[1:3] $rose(ack);
endsequence
assert property (req_ack);
```



an event sequence is described
using a declarative statement

36
SS, SystemVerilog, ModelSim, and You, April 2004

SystemVerilog Assertion Sequences

- SystemVerilog can specify very complex event sequences using a simple and concise syntax
 - Unifies PSL and Verilog syntax to express sequences
 - Adds Verilog-like simulation timing and assertion control
 - Can specify:
 - Advancing one or more clock cycles, using ##
 - Boolean expressions, using special operators
 - and, intersect, or, first_match, throughout, within, \$rose, \$fell, \$stable
 - Repetition of sequences
 - Implication of sequences

37
SS, SystemVerilog, ModelSim, and You, April 2004

Special Test Bench Program Blocks

- Verilog uses modules to model the test bench
 - Modules are intended to model hardware
 - No test semantics to avoid race conditions with the design
- SystemVerilog adds a special program block for testing
 - Events are synchronized to hardware events to avoid races

```
program test (input clk, input [15:0] addr, inout [7:0] data);
  initial begin
    @(negedge clk) data = 8'hC4;
                address = 16'h0004;
    @(posedge clk) verify_results;
  end
  task verify_results;
    ...
  endtask
endprogram
```

- No race conditions between program block and design blocks
- In a module, this example could have race conditions with the design, if the design used the same posedge of clock.

38
SS, SystemVerilog, ModelSim, and You, April 2004

Object Oriented Verification

- SystemVerilog adds “classes” to the Verilog language

- Allows Object Oriented Programming techniques

- Primary intent is for use in verification
- Can be used in abstract h/w models

- Can contain

- “properties” (data declarations)
- “methods” (tasks and functions)

- Similar to C++

- Inheritance
- Public, local or private encapsulation
- New objects created and initialized using new
- Polymorphism

```
class Packet ;
  bit [3:0] command;
  bit [39:0] address;
  bit [4:0] master_id;
  integer time_requested;
  integer time_issued;
  integer status;

  task clean();
    command = 4'h0;
    address = 40'h0;
    master_id = 5'b0;
  endtask

  task issue_request(int delay);
    ...
  endtask
endclass
```

39
SS, SystemVerilog, ModelSim, and You, April 2004

Dynamic Arrays and Associative Arrays

- Verilog has static arrays

- The size of the array is fixed at compile time and cannot change

```
reg [31:0] mem [0:1024];
integer table [0:255];
```

- SystemVerilog adds:

- Dynamic arrays

- The size of the array is left open-ended
- Built-in class methods are used to change the array size during simulation

```
logic [31:0] mem [];
int table [];
```

- Associative arrays

- The index into the array can be non-sequential values
- Built-in class methods are used to access the array

```
typedef enum {A, B, C, D} state;
int table [state];
data = table[A];
```

40
SS, SystemVerilog, ModelSim, and You, April 2004

Enhanced Synchronization: Mailboxes and Semaphores

- **SystemVerilog includes built-in class definitions to synchronize verification activity**
 - **Semaphores**
 - Represents a bucket with a fixed number of keys
 - Built-in class methods used to check keys in and out
 - Process can check out one or more keys, and return them later
 - If not enough keys are available, the process execution stops and waits for keys before continuing (gives mutually exclusive control)
 - **Mailboxes**
 - Represents a FIFO to exchange messages between processes
 - Built-in methods allow adding a message or retrieving a message
 - If no message is available, the process can either wait until a message is added, or continue and check again later

41
SS, SystemVerilog, ModelSim, and You, April 2004

Constrained Random Values

- **Verilog's \$random returns a 32-bit signed random number**
 - No way to constrain the random values returned
- **SystemVerilog adds:**
 - **rand** built-in class for creating distributed random numbers
 - **randc** built-in class for creating cyclic random numbers
 - **SystemVerilog random values can be constrained**

```
class Bus;
  randc bit[15:0] addr;
  rand  bit[31:0] data;

  // constrain addr to be word aligned
  constraint word_align {addr[1:0] == 2'b0;}
endclass
```

42
SS, SystemVerilog, ModelSim, and You, April 2004

What's Next

- ✓ Objectives and caveats
- ✓ Convenience enhancements to Verilog
- ✓ RTL modeling enhancements to Verilog
- ✓ Abstract modeling enhancements to Verilog
- ✓ Verification enhancements to Verilog
- ❑ **ModelSim support for SystemVerilog**
- ❑ Suggestions on adopting SystemVerilog
- ❑ Conclusions

43
SS, SystemVerilog, ModelSim, and You, April 2004

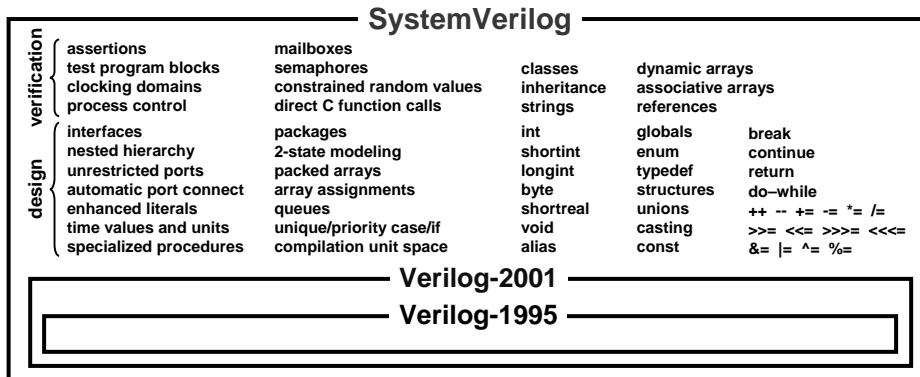
SystemVerilog Roadmap

- **The SystemVerilog standard was developed by Accellera**
 - **June 2001: work began on specification of SystemVerilog**
 - **May 2002: SystemVerilog 3.0 released (modeling extensions)**
 - **May 2003: SystemVerilog 3.1 released (verification extensions)**
 - **March 2004: SystemVerilog 3.1a released (more verification)**
 - **June 2004: SystemVerilog to be donated to IEEE 1364 Verilog**
 - **To be incorporation into next IEEE 1364 standard (2005 or 2006)**
- **Accellera is a consortium of EDA tool vendors and users**
 - **A think tank for developing new EDA standards**
 - **Sponsors IEEE standards groups for Verilog, VHDL, SDF, ...**
 - **Mentor Graphics actively participated in the Accellera SystemVerilog standards committees**

44
SS, SystemVerilog, ModelSim, and You, April 2004

ModelSim and SystemVerilog

- The current Mentor ModelSim simulator (5.8b) supports many of the SystemVerilog constructs presented in this paper
 - Next version will support most of the constructs presented



45
SS, SystemVerilog, ModelSim, and You, April 2004

ModelSim and SystemVerilog

*You can start using
SystemVerilog today!*



- All versions of ModelSim (PE, LE, SE) support the same SystemVerilog constructs
- No special licenses are needed — SystemVerilog is simply the next generation of the Verilog language
 - A `-sv` invocation option is required
 - Enables support for the new keywords added with SystemVerilog

46
SS, SystemVerilog, ModelSim, and You, April 2004

What's Next

- ✓ Objectives and caveats
- ✓ Convenience enhancements to Verilog
- ✓ RTL modeling enhancements to Verilog
- ✓ Abstract modeling enhancements to Verilog
- ✓ Verification enhancements to Verilog
- ✓ ModelSim support for SystemVerilog
- ❑ **Suggestions on adopting SystemVerilog**
- ❑ Conclusions

47
SS, SystemVerilog, ModelSim, and You, April 2004

Categorizing SystemVerilog Extensions

- **This paper has divided SystemVerilog into major categories**
 - **Convenience extensions to the Verilog HDL**
 - Makes Verilog easier to use
 - **RTL modeling extensions to the Verilog HDL**
 - Ensures simulation/synthesis compatibility
 - Built-in checking that models match intent
 - **Abstract modeling extensions to the Verilog HDL**
 - Allows modeling more complex logic in concise, readable code
 - Synthesizable
 - **Verification extensions to the Verilog HDL**
 - Allows writing state-of-the-art verification programs
 - Object oriented test programs
 - Directed testing and constrained random testing

48
SS, SystemVerilog, ModelSim, and You, April 2004

Adopting SystemVerilog

- A good way to get started with SystemVerilog is to...
 - ① Begin using the convenience options right away
 - Makes Verilog easier to use
 - ② Next, start using the RTL extensions
 - Take advantage of built-in checking to ensure models match intent
 - ③ Begin using assertions as soon as possible
 - Key methodology for verifying any size design!
 - ④ Adopt abstract modeling extensions as needed
 - Mostly benefits large multi-million gate designs
 - May require learning new modeling styles and coding tricks
 - May have to wait for EDA tools to implement some extensions
 - ⑤ Plan to adopt advanced verification late 2004 or early 2005
 - Requires adopting object-oriented methodologies
 - Not yet implemented in most EDA tools

49
SS, SystemVerilog, ModelSim, and You, April 2004

What's Next

- ✓ Objectives and caveats
- ✓ Convenience enhancements to Verilog
- ✓ RTL modeling enhancements to Verilog
- ✓ Abstract modeling enhancements to Verilog
- ✓ Verification enhancements to Verilog
- ✓ ModelSim support for SystemVerilog
- ✓ Suggestions on adopting SystemVerilog
- **Conclusions**

50
SS, SystemVerilog, ModelSim, and You, April 2004

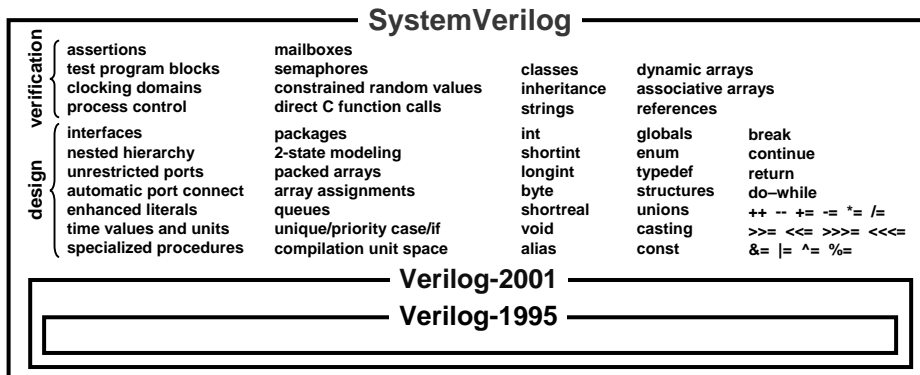
SystemVerilog Extends Verilog

- **SystemVerilog**
 - Is based on the IEEE 1364-2001 standard
 - Adds extensions to make Verilog easier to use
 - Adds extensions to enforce synthesis compatibility
 - Adds extensions to model more logic in fewer lines of code
 - Adds advanced, object-oriented verification capability
- **SystemVerilog is next generation of IEEE Verilog standard**
- **Major EDA companies have already implemented many of the SystemVerilog extensions**
 - Current Mentor ModelSim simulator (5.8b) supports some SystemVerilog extensions
 - Next release will support much of SystemVerilog

51
SS, SystemVerilog, ModelSim, and You, April 2004

SystemVerilog and You

- **SystemVerilog makes Verilog easier, safer and more powerful**
- **There are many SystemVerilog extensions you can and should take advantage of today!**



52
SS, SystemVerilog, ModelSim, and You, April 2004

Some Recommended References

- **SystemVerilog 3.1a Language Reference Manual**
 - 2004, published by Accellera, www.accellera.org
- **The 1364-2001 Verilog HDL Language Reference Manual**
 - 2001, published by the IEEE, www.ieee.org
- **SystemVerilog For Design**
 - Stuart Sutherland, Peter Flake, Simon Davidmann & Phil Moorby, 2004, Kluwer, ISBN: 0-4020-7530-8
- **Verilog-2001: A Guide to the New Features in the Verilog Hardware Description Language**
 - Stuart Sutherland, 2001, Kluwer, ISBN: 0-7923-7568-8

53
SS, SystemVerilog, ModelSim, and You, April 2004

About the Presenter...

- **Stuart Sutherland**
 - Verilog design consultant, specializing in Verilog training
 - Hardware design engineer with a Computer Science degree
 - Deeply involved with Verilog since 1988
 - Member of IEEE 1364 Verilog standards group since 1993
 - Co-chair of Verilog PLI task force
 - Technical editor of PLI sections of the IEEE 1364 Verilog Language Reference Manual
 - Member of the Accellera committee defining SystemVerilog
 - Involved since the inception of the SystemVerilog standardization
 - Technical editor of SystemVerilog Reference Manual

54
SS, SystemVerilog, ModelSim, and You, April 2004