Awarded Best Paper Presentation for SNUG Silicon Valley 2013 ☺

Synthesizing SystemVerilog

Busting the Myth that SystemVerilog is only for Verification

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

What This Paper is About...

- ✓ Debunking a myth regarding SystemVerilog
- ✓ What constructs in SystemVerilog are synthesizable
- ✓ Why those constructs are important for you to use
- ✓ How well Design Compiler and Synplify-Pro support SystemVerilog synthesis
- ✓ Fifteen coding recommendations for getting the most from Synthesizable SystemVerilog

Only a few Synthesizable SystemVerilog constructs are discussed in this presentation; Refer to the paper for the full list and details of Synthesizable SystemVerilog

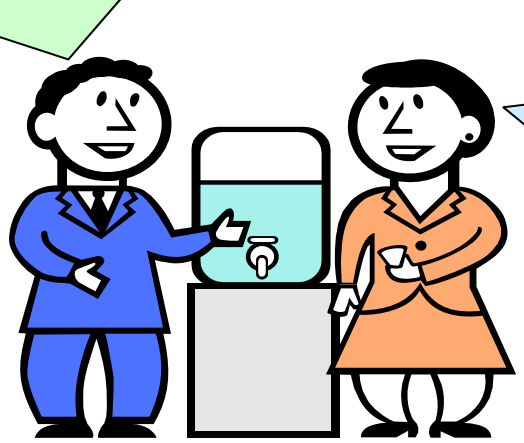
Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

It's a Myth!

Verilog is a design language, and SystemVerilog is a verification language



And synthesis compilers can't read in SystemVerilog

- ***Not True!*** – SystemVerilog was designed to enhance both the *design and verification* capabilities of traditional Verilog
- Technically, there is no such thing as "*Verilog*" – the IEEE changed the name to "*SystemVerilog*" in 2009
- VCS, Design Compiler and Synplify-Pro all support RTL modeling with SystemVerilog

Much of SystemVerilog is Intended to be Synthesizable

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

SystemVerilog-2005/2009/2012

verification

assertions	mailboxes	classes	dynamic arrays	2-state types
test program blocks	semaphores	inheritance	associative arrays	shortreal type
clocking domains	constrained random values	strings	queues	globals
process control	direct C function calls	references	checkers	let macros

design

interfaces	packed arrays	break	enum	++ -- += -= *= /=
nested hierarchy	array assignments	continue	typedef	>>= <<= >>>= <<<=
unrestricted ports	unique/priority case/if	return	structures	&= |= ^= %=
automatic port connect	void functions	do–while	unions	==? !=?
enhanced literals	function input defaults	case inside	2-state types	inside
time values and units	function array args	aliasing	packages	streaming
specialized procedures	parameterized types	const	$unit	casting

Verilog-2005

uwire	`begin_keywords	`pragma	$clog2

Verilog-2001

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

Verilog-1995 (created in 1984)

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

Synopsys Users Group
SILICON VALLEY 2013

Part One:
SystemVerilog Declaration Enhancements

The Goal…

- Model more functionality in fewer lines of code
- Reduce redundancy
- Reduce the risk of coding errors

New Synthesizable Variable Data Types

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

- Useful synthesizable variable types
  - **`logic`** — 4-state variable, user-defined size (replaces **`reg`**)
  - **`enum`** — a variable with a specified set of legal values
  - **`int`** — 32-bit 2-state var (use with for-loops, replaces **`integer`**)

- **What's the advantage?**

  - ✓ **`logic`** makes code more self-documenting (**`reg`** does not infer a "register," but it looks like it does)
  - ✓ The **`enum`** type is important – more on another slide

- Other synthesizable variable types … not very useful in RTL
  - **`bit`** — single bit 2-state variable
  - **`byte`** — 8-bit 2-state variable
  - **`shortint`** — 16-bit 2-state variable
  - **`longint`** — 64-bit 2-state variable

Although synthesizable, these types are best used in testbenches

Avoid 2-state types in synthesizable models – they can hide serious design bugs!

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Simplified Port Type Rules

- Traditional Verilog has strict and confusing rules for port types
  - Input ports must be a net type (**`wire`**)
  - Output ports must be:
    - **`reg`** (a variable) if assigned from a procedural block (initial, always)
    - **`wire`** if assigned from a continuous assignment
    - **`wire`** if driven by an instance of a module or primitive output

```
module chip
 (input  wire  in1,
  input  wire  in2,
  output reg   out1,
  output wire  out2
 );
```

- SystemVerilog makes it easy…
  - Just declare everything as **`logic`** !!!

```
module chip
 (input  logic in1,
  input  logic in2,
  output logic out1,
  output logic out2
 );
```

"`logic`" indicates the value set (4-state) to be simulated – SystemVerilog infers a variable or net based on context

- **What's the advantage?**

  - ✓ Creating and modifying modules just got a whole lot easier!

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Enumerated Types

- SystemVerilog adds enumerated types to Verilog
  - **enum** defines variables or nets with a legal set of values
    - Each legal value is represented by a label

```
enum logic [2:0] {WAIT=3'b001, LOAD=3'b010, READY=3'b100} state;
```

- Enumerated types have strict rules
  - The label value must be the same size as the variable
  - Can be assigned a label from the enumerated list
  - Can be assigned the value of an identical enumerated variable
  - All other assignments are illegal

- **What's the advantage?** 
  - ✓ Enumerated types can prevent inadvertent (and hard to debug) coding errors (example on next slide)

Synopsys Users Group
SILICON VALLEY 2013

The Advantage of Enumerated Variables

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Traditional Verilog

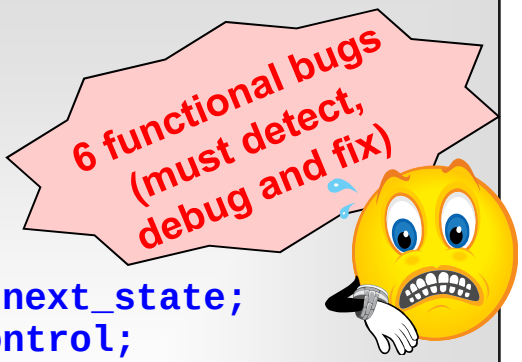
```
parameter [2:0]
  WAIT = 3'b001,
  LOAD = 3'b010,
  DONE = 3'b001;
parameter [1:0]
  READY = 3'b101,
  SET   = 3'b010,
  GO    = 3'b110;

reg [2:0] state, next_state;
reg [2:0] mode_control;

always @(posedge clk or negedge rstN)
  if (!resetN) state <= 0;
  else         state <= next_state;

always @(state)  // next state decoder
  case (state)
    WAIT : next_state = state + 1;
    LOAD : next_state = state + 1;
    DONE : next_state = state + 1;
  endcase

always @(state)  // output decoder
  case (state)
    WAIT : mode_control = READY;
    LOAD : mode_control = SET;
    DONE : mode_control = DONE;
  endcase
```



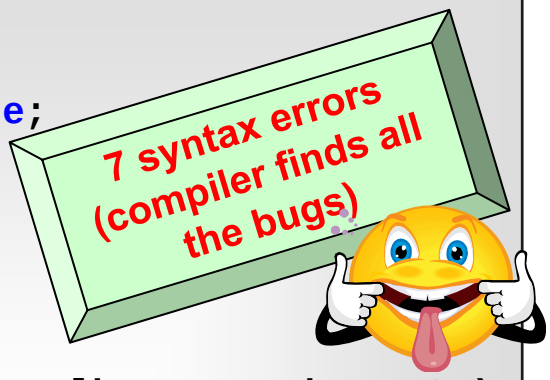
SystemVerilog

```
enum logic [2:0]
  {WAIT = 3'b001,
   LOAD = 3'b010,
   DONE = 3'b001}
  state, next_state;
enum logic [1:0]
  {READY = 3'b101,
   SET   = 3'b010,
   GO    = 3'b110}
  mode_control;

always_ff @(posedge clk or negedge rstN)
  if (!resetN) state <= 0;
  else         state <= next_state;

always_comb  // next state decoder
  case (state)
    WAIT : next_state = state + 1;
    LOAD : next_state = state + 1;
    DONE : next_state = state + 1;
  endcase

always_comb  // output decoder
  case (state)
    WAIT : mode_control = READY;
    LOAD : mode_control = SET;
    DONE : mode_control = DONE;
  endcase
```



Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Structures

- SystemVerilog structures bundle multiple variables together
  - The entire structure can be assigned a list of values
  - Entire structure can copied to another structure of same type
  - Entire structures can be passed through module ports

```
struct {
  logic [ 7:0] opcode;
  logic [31:0] data;
  logic        status;
} operation;
```

```
operation = '{8'h55, 1024, 1'b0};
```

Assign entire structure

```
operation.data = 32'hFEEDFACE;
```

Assign to structure member

- **What's the advantage?**

  - ✓ Bundle related signals together under one name
  - ✓ Reduce lines of RTL code substantially
  - ✓ Reduce risk of declaration mismatches
  - ✓ Can eliminate design errors often not found until late in a design cycle (inter-module mismatches, missed assignments, ...)

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

User-defined Types

- SystemVerilog adds user-defined types to Verilog
  - **typedef** defines a new type
    - Can be based on built-in types or other user-defined types
    - Variables and nets can be declared as a user-defined type

```
typedef logic [31:0] bus32_t;
typedef enum [7:0] {ADD, SUB, MULT, DIV, SHIFT, ROT, XOR, NOP} opcodes_t;
typedef enum logic {FALSE, TRUE} boolean_t;

typedef struct {
  opcodes_t  opcode;
  bus32_t    data;
  boolean_t  status;
} operation_t;
```

- **What's the advantage?**
  - ✓ Can define complex types once and use many times
  - ✓ Ensures consistency throughout a module

```
module ALU (input  operation_t operation,
            output bus32_t     result);
  operation_t registered_op;
  ...
endmodule
```

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Packages

- SystemVerilog adds a package construct to Verilog
  - Allows the same definition to be used by many modules

```
package project_types;
  typedef logic [31:0] bus32_t;
  typedef enum [7:0] {...} opcodes_t;
  typedef struct {...} operation_t;
  function automatic crc_gen ...;
endpackage
```

```
module ALU
  import project_types::*;
  (input  operation_t operation,
   output bus32_t     result);
  operation_t registered_op;
  ...
endmodule
```

- **What's the advantage?**

  - ✓ Ensures consistency throughout a project (including verification)
  - ✓ Reduces duplicate code
  - ✓ Makes code easier to maintain and reuse than `include
  - ✓ Controlled scope

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

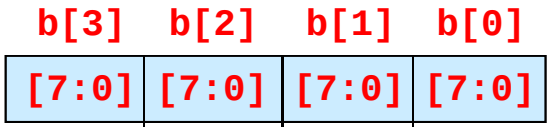
Synopsys Users Group
SILICON VALLEY 2013

Data Arrays

- Packed array (aka "vector") enhancements
  - Vectors can now be divided into sub fields

```
logic [3:0][7:0] b;
```

a 32-bit vector with 4 8-bit subfieds



- Unpacked array enhancements
  - Can now have arrays of structures, user-defined types, etc.
  - C-like array declarations
  - Assign to entire array at once
  - Copy arrays
  - Pass arrays through ports

```
logic [7:0] a1 [0:1][0:3];
logic [7:0] a2 [2][4];

a1 = '{'{7,3,0,5},'{default:'1}};

a2 = a1;
```

C-like declaration

assign values to entire array

copy entire array

- **What's the advantage?**
  - ✓ **This is major!** – Manipulating entire data arrays substantially reduces lines of code *(see example on next page)*

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Working with Entire Arrays Reduces Lines of Code

```
package design_types;
  typedef struct {
    logic [ 3:0] GFC;
    logic [ 7:0] VPI;
    logic [15:0] VCI;
    logic        CLP;
    logic [ 2:0] T;
    logic [ 7:0] HEC;
    logic [ 7:0] Payload [48];
  } uni_t; // UNI cell definition
endpackage
```

This structure bundles 54 variables together (including the array of 48 Payload variables)

54 ports in old Verilog

another 54 ports

```
module transmit_reg (output design_types::uni_t  data_reg,
                     input  design_types::uni_t  data_packet,
                     input  logic                clock, resetN);
  always @(posedge clock or negedge resetN)
    if (!resetN) data_reg <= '{default:0};
    else         data_reg <= data_packet;
endmodule
```

54 separate assignment statements in old Verilog

54 more separate assignment statements in old Verilog

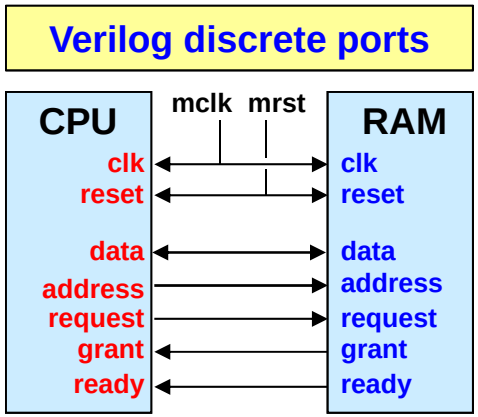
- **What's the advantage?**
  - ✓ 4 lines of code in SystemVerilog replaces 216 lines of old Verilog – and ensures consistency in all 4 places!

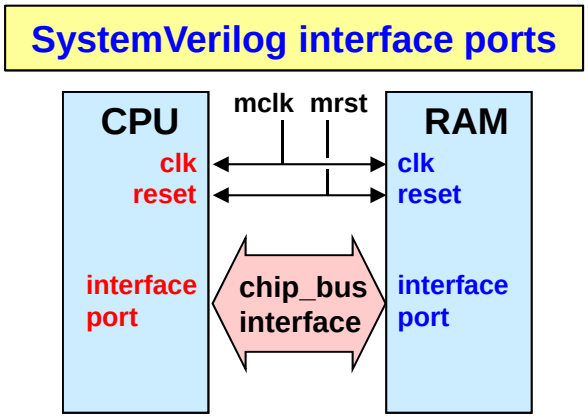
Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Interface Ports

- SystemVerilog interfaces are a compound, multi-signal port
  - Bundles any number of signals (nets and variables) together
  - Bundles "methods" (tasks and functions) with the signals
  - Bundles assertion checks with the signals






```
interface chip_bus;
  logic [31:0] data, address;
  logic        request, grant,
  boolean_t    ready;
endinterface

module CPU (chip_bus bus,
            input logic clk,
            input logic reset);
  ...
```

- **What's the advantage?**

  - ✓ Simplifies complex bus definitions and interconnections
  - ✓ Ensures consistency throughout the design

Synopsys Users Group
SILICON VALLEY 2013

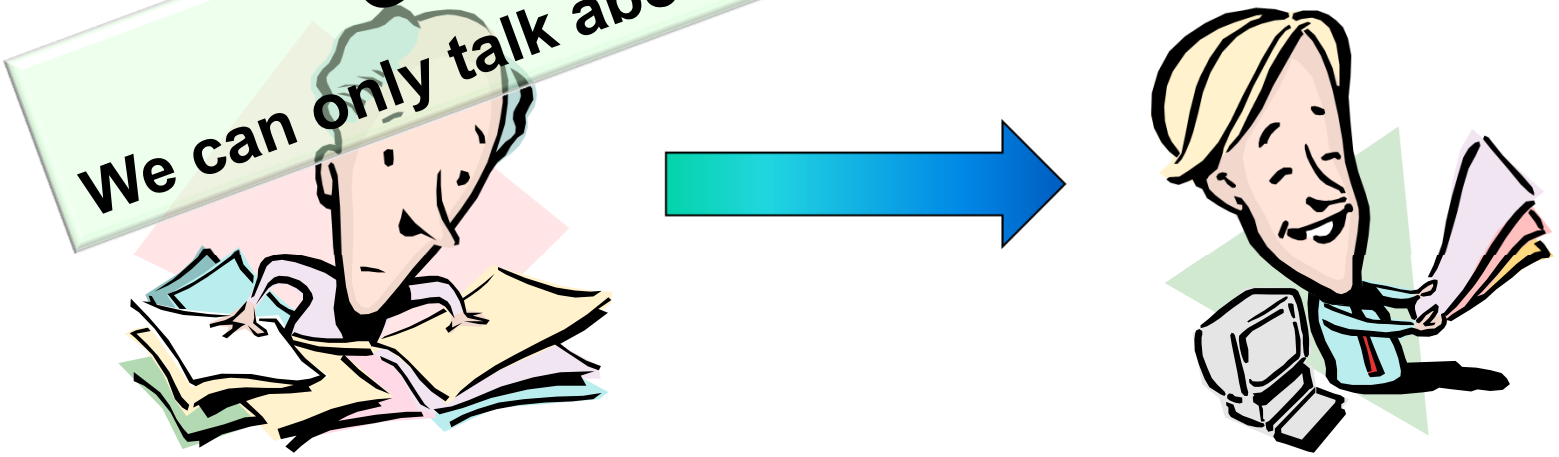
Part Two:
SystemVerilog Programming Enhancements

The Goal…

- Model RTL functionality more accurately
- Reduce mismatches in RTL simulation vs. synthesized gates
- Fewer lines of code – concisely model complex functionality

We can only talk about a few features in this presentation

Go to the Paper for full details!



Hardware Specific Procedural Blocks

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

- SystemVerilog adds special hardware-oriented procedures: **`always_ff`**, **`always_comb`**, and **`always_latch`**
  - Document engineer's intent
  - Software tool can verify that functionality meets the intent
  - Enforce several semantic rules required by synthesis

```
always @(mode)
  if (!mode)
    o1 = a + b;
  else
    o2 = a - b;
```

Traditional Verilog
Synthesis must guess (infer) what type of logic was intended

```
always_comb
  if (!mode)
    o1 = a + b;
  else
    o2 = a - b;
```

SystemVerilog
Contents checked for adherence to synthesis rules for combinational logic

- **What's the advantage?**

- ✓ RTL code intent is self-documented
- ✓ Non-synthesizable code won't simulate
- ✓ Simulation, synthesis and formal tools use same rules

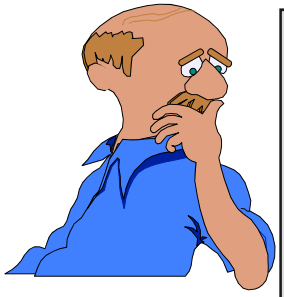
Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

The case() inside Decision Statement

- The **case() inside** statement replaces **casex** and **casez**
  - Bits set to X, Z or ? *in the case items* are "don't care" bits
  - Any X, Z or ? bits in the *case expression* are *not* don't cares
    - With **casez** and **casex**, X, Z of ? bits in the case expression are also considered don't cares – which is a serious problem



```
case (opcode) inside
  8'b1???????: ... // only compare most significant bit
  8'b????1111: ... // compare lower 4 bits, ignore upper bits
  ...
  default: $error("bad opcode");
endcase
```

If opcode has the value 8'bzzzzzzzz, which branch should execute?

- **What's the advantage?**

  - ✓ **case() inside** eliminates the serious ***GOTCHA*** of **casex** and **casez** than could lead to design bugs going undetected

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Unique and Priority Decisions

- The **`unique`**, **`unique0`** and **`priority`** decision modifiers…
  - Enable **`parallel_case`** and/or **`full_case`** synthesis pragmas
  - Enable run-time simulation checking for when the decision might not work as expected if synthesized with the pragma

```
always_comb
  unique case (state)
    RDY: ...
    SET: ...
    GO : ...
  endcase
```

- **Enables full_case and parallel_case pragmas**
- **Will get simulation warnings if `state` matches multiple branches (not a valid parallel_case)**
- **Will get simulation warnings if `state` doesn't match any branch (not a valid full_case)**

- **What's the advantage?**

  - ✓ Automatic run-time checking that the decision statement will synthesize as intended

***WARNING*: These decision modifiers do not eliminate the evil side of the full_case and parellel_case twins –– but, the keywords do warn about the presence of evil**

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Operators

- SystemVerilog adds many new synthesizable constructs:
  - **==?** and **!=?** wildcard equality/inequality operators
  - **inside** set membership operator
  - **<<**, **>>** pack and unpack streaming operators
  - **++** and **--** increment and decrement operators
  - **+=**, **-=**, ***=**, **/=** … assignment operators

```
if (data inside {[0:255]}) ...
```
if data is between 0 to 255, inclusive

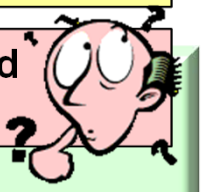
```
if (data inside {3'b1?1}) ...
```
if data is 3'b101, 3'b111, 3'b1x1, or 3'b1z1

```
a = { << { b }};
```
bit reverse – unpack bits of b and assign to a in reverse order

```
c = { <<8{ d }};
```
byte reverse – unpack 8-bit chunks of d and assign in reverse order

- **What's the advantage?**

How much Verilog code would these operations require?



  - ✓ Model more RTL functionality in fewer lines of code

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Type, Size and Sign Casting

- SystemVerilog adds casting operations to Verilog
  - <type>'(<expression>) — cast expression to different data type
  - <size>'(<expression>) — casts expression to a vector size
  - signed'(<expression>) — casts expression to signed
  - unsigned'(<expression>) — casts expression to unsigned

```
logic [31:0] a, y;
logic [ 5:0] b;
y = {a,a} >> b;
```

Rotate a by b number of times

Will get warning from lint checkers and synthesis because LHS is 32 bits and RHS is 64 bits

```
y = logic [31:0]'({a,a} >> b);
```

cast the operation result to 32 bits so that the RHS and the LHS are the same size

- **What's the advantage?**

  - ✓ Documents intent that a change in type, size or sign is intended
  - ✓ Can eliminate size and type mismatch warnings

Module Instance Port Connection Shortcuts

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

- Verilog netlist port connections must name both the port and the net connected to it

can be verbose and redundant

```
module dff (output q, qb,
            input  clk, d, rst, pre);
  ...
```

```
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]));
```

- 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 with the same name

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

- **What's the advantage?**

  - ✓ Reduce typing (and typos) when connecting design blocks
  - ✓ Built-in checking prevents connection mismatches

Enhanced Literal Value Assignments

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

- In Verilog, there is no simple way to fill a vector with all 1's

```
parameter N = 64;
reg [N-1:0] data_bus;
data_bus = 64'hFFFFFFFFFFFFFFFF; //set all bits of data_bus to 1
```

could also use coding tricks, such as replicate or invert operations

vector width must be hard coded

- SystemVerilog adds a vector fill literal 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 [N-1:0] data_bus;
data_bus = '1;
```

set all bits of data_bus to 1

- **What's the advantage?**

  - ✓ Code will scale correctly when vector sizes change
  - ✓ Don't need to know obscure coding tricks such as replicate

Verilog and SystemVerilog Compatibility Directives

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

- SystemVerilog is backward compatible with Verilog
  - Old Verilog and SystemVerilog models can be intermixed
- SystemVerilog does add many keywords to Verilog
  - In Verilog models, those keywords were legal to use as names
  - The `begin_keywords directive tells software tools which version of reserved keywords to use during compilation

```
`begin_keywords 1364-2001
module test;
  wire priority;
  ...
endmodule
`end_keywords
```

In Verilog "priority" *is not* a reserved keyword

```
`begin_keywords 1800-2005
module decoder (...);
  always_comb
    priority case (...);
    ...
endmodule
`end_keywords
```

In SystemVerilog "priority" *is* a reserved keyword

- **What's the advantage?**

  - ✓ Ensures design code is reusable, past, present and future

Lots of Enhancements to Tasks and Functions

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

- SystemVerilog enhancements tasks and functions several ways
  - Void functions – *this one is important for synthesis!*
  - Functions with output and inout formal arguments
  - Formal arguments default to input
  - Arrays, structures, user-defined types as formal arguments
  - Pass by name in task/function calls
  - Function return values can be specified, using return
  - Parameterized task/function arguments using static classes

See the paper for details

- **What's the advantage?**
  - ✓ Fewer lines of code
  - ✓ Reusable code

Recommendation – use void functions instead of tasks in synthesizable models

Synopsys Users Group
SILICON VALLEY 2013

Part Three:
Synthesis Considerations

The paper also discusses…

- Design Compiler versus Synplicity-Pro
- Some things that should be synthesizable
- 15 recommendations for how *you* can benefit from SystemVerilog

Differences between Design Compiler and Synplicity-Pro

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

- DC and Synplify-Pro are closely aligned, but there are some differences in the SystemVerilog constructs supported

SystemVerilog Construct	Design Compiler 2012.06-SP4	Synplify-Pro 2012.09
`‘begin_keyword`, `‘end_keyword` compatibility directives	yes	no
Package `import` before module port list	yes	no
`case`...`inside`	yes	no
`priority`, `unique0` and `unique` modifier to `if`...`else`	yes	ignored
Parameterized tasks and functions (using classes)	yes	no
`real` data type	no	yes
Nets declared from `typedef struct` definitions	no	yes
Immediate assertions	ignored	yes
Interface `modport` expressions	no	yes

See the paper for details

Several important differences are listed in this table – refer to the paper for a more complete list of differences

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

DC and/or Synplicity-Pro Wish List

- SystemVerilog has several constructs that are useful for modeling hardware, but which are not synthesizable
  - uwire single source nets
  - foreach loops
  - Task/function inputs with default values
  - Task/function ref arguments
  - Set membership operator (inside) with expressions
  - Package chaining
  - Extern module declarations
  - Configurations
  - Generic and user-defined net types

See the paper for details

Let your Synopsys rep know if any of these features would help you in your projects!

Fifteen Ways You Can Benefit from Using SystemVerilog in RTL designs

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

1. Use **`logic`** for modules ports and most internal signals – forget **`wire`**, **`reg`**
2. Use the **`uwire`** net type to check for and enforce single-driver logic
3. Use enumerated types for variables with limited legal values
4. Use structures to collect related variables together
5. Use user-defined types to ensure consistent declarations in a design
6. Use packages for declarations that are shared throughout a design
7. Use **`always_comb`**, **`always_latch`** and **`always_ff`** procedural blocks
8. Use **`case`**...**`inside`** instead of **`casez`** and **`casex`**
9. Use **`priority`**, **`unique0`**, **`unique`** instead of **`full_case`**, **`parallel_case`**
10. Use **`priority`**, **`unique0`**, **`unique`** with **`if`**...**`else`** when appropriate
11. Use **`void function`** instead of task in RTL code
12. Use dot-name and dot-star netlist shortcuts
13. Use interfaces to group related bus signals
14. Use **`` `begin_keywords``** to specify the language version used
15. Use a locally declared **`timeunit`** instead of **`` `timescale``**

Stu Sutherland
Sutherland HDL

Don Mills
Microchip

Synopsys Users Group
SILICON VALLEY 2013

Summary

- It's a myth – SystemVerilog is not just for verification, it is also a synthesizable design language
  - Technically, there is no such thing as "*Verilog*" – the IEEE changed the name to "*SystemVerilog*" in 2009
- SystemVerilog adds many important synthesizable constructs to the old Verilog language
  - Design more functionality in fewer lines of code
  - Ensure RTL code will synthesize to the logic intended
  - Make code more reusable in future projects
- Design Compiler and Synplify-Pro both support SystemVerilog
  - There are some differences (see the paper for details)
- There are many benefits to using SystemVerilog for ASIC and FPGA design

Questions?

the answer is in the paper ... somewhere

(if not, we'll find out ☺)

Stu Sutherland
stuart@sutherland-hdl.com

Don Mills
mills@microchip.com
mills@lcdm-eng.com