

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon



*a proposal for a standard
SystemVerilog Synthesis Subset*

Stuart Sutherland
Sutherland HDL, Inc.



www.sutherland-hdl.com

© 2005, Sutherland HDL, Inc.

About the Presenter...



- **Stuart Sutherland, a SystemVerilog wizard**
 - Independent Verilog/SystemVerilog consultant and trainer
 - Hardware design engineer with a Computer Science degree
 - Has been working with Verilog since 1988
 - **Specializing in Verilog and SystemVerilog training**
 - Member of the IEEE 1800 SystemVerilog standards group
 - Involved with the definition of SystemVerilog since its inception
 - Technical editor of SystemVerilog Language Reference Manual
 - Member of IEEE 1364 Verilog standards group since 1993
 - Past chair of the Verilog PLI task force
 - Technical editor of the IEEE 1364-1995, 1364-2001 and 1364-2005 Verilog Language Reference Manuals

www.sutherland-hdl.com

© 2006, Sutherland HDL, Inc.

2 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

What This Paper Will Cover



- Why we need a standard for synthesis
 - What major EDA vendors can synthesize
 - What problems this paper solves
- Synthesizable SystemVerilog Constructs
 - There's a lot!
- Recommendations
 - What EDA companies ought to do
- Summary

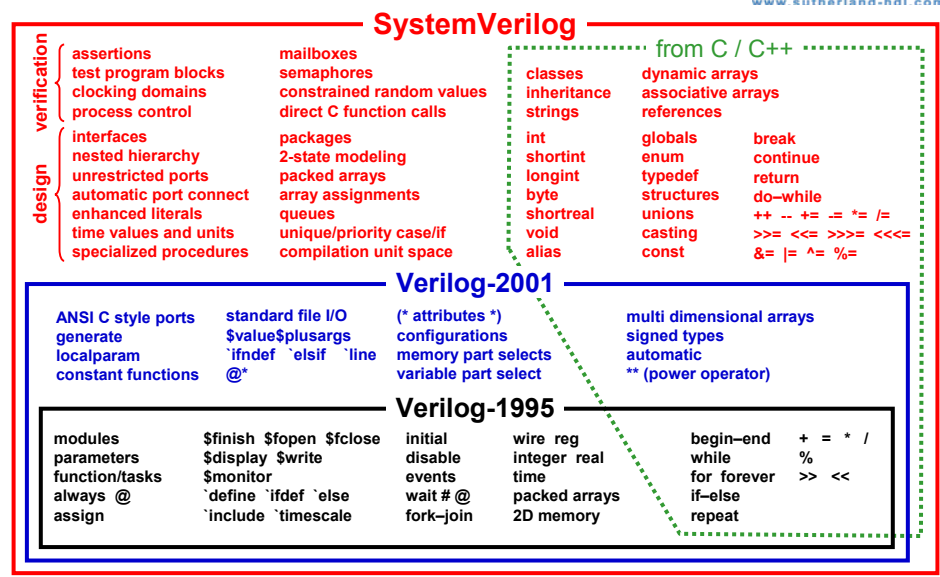


© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

3 of 30

SystemVerilog is a Major Extension to Verilog



A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

Why We Need A SystemVerilog Synthesis Standard



- Most design models must be written to work with simulation tools and synthesis compilers
- Verilog and SystemVerilog are simulation-oriented standards
 - Define event-driven simulation semantics and language rules
 - Define constructs for design, verification and APIs
 - Do not define semantics and rules for synthesis
- Verilog has an IEEE synthesis standard (1364.1)
- The IEEE has not defined a SystemVerilog synthesis standard
 - EDA vendors must define their own synthesis subset and rules
 - Designers get to use trial and error to find a synthesis subset that will work with multiple tools

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

5 of 30

About This Proposal For A SystemVerilog Synthesis Subset



- This Paper defines a portable SystemVerilog Synthesis subset
 - Developed by Sutherland HDL for use in our consulting and training services
 - Reviewed by synthesis R&D groups at major EDA companies
 - Synopsys, Cadence, Mentor Graphics, Synplicity, Magma
 - Only constructs that are universally supported are listed in this proposed synthesis subset
 - Most constructs presented in this paper can be synthesized by all these products today
- The purpose of this paper is to enable engineers to write synthesizable code that is not dependent on specific tools
- The paper also includes some recommendations for expanding the synthesizable subset

© 2006, Sutherland HDL, Inc.


DVCon-2006: Proposed SystemVerilog Synthesis Subset

6 of 30


A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon


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


Let's Look At The Details!



- All we have time for is a quick summary of the proposed subset...
 - Read the paper for more details

Only synthesizable enhancements are listed — if we don't talk about it in the paper, it is probably not synthesizable by most tools

© 2006, Sutherland HDL, Inc.DVCon-2006: Proposed SystemVerilog Synthesis Subset7 of 30


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

Packages

- SystemVerilog adds a package construct to Verilog
 - Allows the same definition to be used by multiple design modules
- The synthesizable constructs for packages are:
 - parameter constant definitions
 - const variable definitions
 - typedef user-defined types
 - Automatic task/function definitions
 - Import statements to import from other packages
- Package definitions can be imported 4 ways — all synthesizable
 - Explicit references of package items
 - Explicit import of package items
 - Implicit “wildcard” import of package
 - Explicit or implicit import into the \$unit space (see next page)

```
package defs;
parameter MAX_SIZE = 128;
typedef enum {s1,s2,s3} states_t;
task automatic fetch (...);
...
endtask
endpackage
```

© 2006, Sutherland HDL, Inc.DVCon-2006: Proposed SystemVerilog Synthesis Subset8 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

The \$unit Compilation Unit Package



- SystemVerilog provides a built-in package called \$unit
 - Any declaration outside of modeling blocks (module, interface or program) is in the \$unit package
 - Automatically imported into all blocks that compiled at the same time
 - Each invocation of a compiler creates a new, unique \$unit package
- Synthesis supports the same constructs in \$unit as with packages

```
parameter MAX_SIZE = 128;
typedef enum {s1,s2,s3} states_t;

module fsm (...);
  states_t state, next_state; // automatic import from $unit
  ...
endmodule
```

CAUTION — Since each invocation of a compiler creates a unique \$unit package, the same files must be compiled together in both simulation and synthesis in order to get the same results!

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

9 of 30

SystemVerilog Variables



- The SystemVerilog variable types that are synthesizable are:
 - **bit** — single bit 2-state variable
 - **logic** — single bit 4-state variable (replaces Verilog **reg** type)
 - **byte** — 8-bit 2-state variable
 - **shortint** — 16-bit 2-state variable
 - **int** — 32-bit 2-state variable
 - **longint** — 64-bit 2-state variable

Caution:

- The 2-state simulation semantics are not preserved by synthesis
 - 2-state variables become 4-state wires in post synthesis netlists
 - Can cause differences in RTL versus gate-level simulation
 - Can affect equivalence checking of RTL versus gate-level models

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

10 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

The var Keyword



- SystemVerilog allows all variables to be declared with an optional **var** keyword

- Used to explicitly document that a declaration is a variable
- Does not change the variable semantics in any way

```
var logic [7:0] a; // 8-bit, 4-state vector variable
```

```
typedef enum bit {FALSE, TRUE} bool_t;  
var bool_t b; // variable of a user-defined type
```

- A variable can also be declared using **var** without an explicit type
 - Defaults to a **logic** type

```
var [7:0] c; // 8-bit logic variable
```

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

11 of 30

SystemVerilog Nets



- Verilog has simple 1-bit net types (**wire**, **tri**, **wand**, **wor**, ...)
- SystemVerilog allows nets to be declared with complex types

```
wire logic [7:0] w1;
```

8-bit 4-state net

```
wire integer w2;
```

32-bit 4-state net

```
typedef enum bit {FALSE, TRUE} bool_t;  
wire bool_t w3;
```

net with bool_t values

```
typedef struct {logic a, integer b} packet_t;  
wire packet_t w4;
```

compound net with subfields

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

12 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

User-defined Types



- SystemVerilog adds several forms of user-defined data types
 - The synthesizable user-defined types are **typedef** and **enum**
- typedef** defines a new type based on **built-in types** or other **user-defined types** (similar to C)
- enum** defines variables or nets with a **specified set of values**
 - Can be a simple enumerated type

```
enum {WAITE, LOAD, READY} state;
```

- Defaults to a base data type of **int**
- The first label defaults to a value of **0**
- Subsequent labels increment from the previous label value

- Can specify an explicit base type
- Can specify explicit values for each label

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

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

13 of 30

Structures



- Structures provide a mechanism to collect multiple variables together under a common name

```
struct {  
    logic [1:0] parity;  
    logic [47:0] data1;  
    logic [63:0] data2;  
} data_word_s;
```

- Structure variables can be assigned individually

```
data_word_s.data1 = 48'hF;
```

- The entire structure can be assigned a list of values

```
data_word_s = '{2'b10, 55, 1024};
```

- Structures can be “packed”, making them a vector with subfields
 - Can be used as a vector in operations

```
typedef struct packed {  
    logic [7:0] tag;  
    int a, b;  
} packet_t;
```

```
var packet_t data_in, data_out, data_check;  
assign data_check = data_in ^ data_out;
```

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

14 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

Unions



- Unions treat a single storage space as several different data types

```
union tagged {  
  logic [23:0] short_word;  
  logic [63:0] long_word;  
} data_word_u;
```

```
data_word_u.short_word = 24'hF; // write to union
```

```
data = data_word_u.short_word; // read from union
```

- SystemVerilog has three types of unions:
 - Simple unions:** can store any data type — **not synthesizable**
 - Packed unions:** all types represented must be the same total number of bits — **synthesizable**
 - Tagged unions:** can store any types, but must read from the same type as the last write — **synthesizable**
 - Simulation maintains an internal “tag” to check if the data type read matches the data type used the last time a value was written
 - Synthesis does not create the implied tag in the hardware implementation

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

15 of 30

Data Arrays



- Packed arrays** (Verilog vectors)
 - Vectors can be divided into sub fields

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

a[3]	a[2]	a[1]	a[0]
[7:0]	[7:0]	[7:0]	[7:0]

- Unpacked arrays** (Verilog arrays)
 - Arrays of structures, user-defined types, etc.
 - C-like array declarations
 - Initialize arrays
 - Copy arrays
 - Assign list of values to arrays
 - Assign to slices of an array
 - Pass arrays through module ports
 - Pass arrays to/from tasks and functions
 - Array traversal **foreach** loop
 - Array query functions: **\$dimensions**, **\$left**, **\$right**, **\$low**, **\$high**, **\$increment**, **\$size**

```
int a1 [2][4]; //C-like declaration  
int a2 [0:1][0:3] = '{default:'1';  
  // initialize array at declaration  
  
a1 = a2; // copy array  
  
a2 = '{ '{7,3,0,5}', '{2,0,1,6}}';  
  // assign list of values
```

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

16 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

Type, Size and Sign Casting



- SystemVerilog adds casting operations to the Verilog language
 - Casting follows the same rules as an assignment statement
 - <type>' (<expression>)** — cast expression to different data type

```
int a, b, y;
shortreal r;
y = a + int'(r ** b); //cast operation result to int
```

- <size>' (<expression>)** — casts expression to a vector size

```
bit [15:0] a, b, y;
y = a + b**16'(2); //cast literal value 2 to be 16 bits wide
```

- <sign>' (<expression>)** — casts expr. to **signed** or **unsigned**

```
shortint a, b;
int y;
y = y - signed'{a,b}; //cast concatenation result to signed value
```

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

17 of 30

Port Declarations



- SystemVerilog relaxes the overly-restrictive Verilog port rules
 - Internal data type of input ports can be a variable
 - Arrays and array slices can be passed through ports
 - Typed structures, unions, and user-defined types can be passed

```
package user_types;
  typedef enum bit (FALSE, TRUE) bool_t;
endpackage

typedef struct { // declared in $unit space
  logic [31:0] i0, i1;
  logic [ 7:0] opcode;
} instruction_t;

module ALU (output logic table [0:3][0:7], // array port
           output user_types::bool_t ok, // enum port
           input instruction_t IW ); // structure port
```

- Beware of port mangling!** Synthesis may convert compound ports:
 - to a **single vector** that is a concatenation of the compound port
 - or**, to **separate ports** for each part of a compound port

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

18 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

Hardware Specific Procedural Blocks



- The Verilog **always** procedural block is general purpose
 - Used to model **combinational**, **latched**, and **sequential logic**
 - Synthesis must “infer” (*guess*) the type of hardware intended
- SystemVerilog adds special hardware-oriented procedures: **always_ff**, **always_comb**, and **always_latch**
 - Enforce several semantic rules required by synthesis
 - Simulation, synthesis and formal tools to use same rules
 - Tools can check that designer’s intent has been modeled

```
always_comb  
{  
  if (!mode)  
    y = a + b;  
  else  
    y = a - b;  
}
```

sensitivity list automatically inferred

contents must follow several of the synthesis requirements for combinational logic

Synthesis compilers can warn if the functionality does not match the designer’s intent for combinational logic!

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

19 of 30

Operators and Programming Statements



- SystemVerilog adds several synthesizable operators:
 - ++ and -- increment and decrement
 - +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, <<<=, >>>= assignment
 - ==? and !=? wild equality/inequality operators (similar to **casex**)

```
if (address ==? 16'hFF??) // lower 8 bits ignored
```

- Synthesizable enhancements to Verilog programming statements:
 - Variables on left-hand side of continuous assignments
 - Multiple loop control variables in for loops
 - **foreach** array traversal loop
 - **do...while** loops
 - **break**, **continue** and **return** jump statements
 - **unique** and **priority** decision statements (see next two slides)

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

20 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

The Unique Decision Modifier



- The **unique** decision modifier...
 - Indicates that the decisions *may* be evaluated in parallel
 - Simulation and synthesis warn if detect overlap in the decisions
 - Simulation has run-time warning if there is no matching condition
 - For synthesis, **unique** is equivalent to the combined pragmas:
`//synthesis parallel_case full_case`

given the declaration: `bit [1:0] a;`

```
unique case (a)
  0, 1: ...
  2   : ...
endcase
```

- Simulation warns if "a" has a value that is not decoded
- Synthesis ignores unspecified values (full_case)

```
unique if (a == 0) ...
else if (a == 2) ...
```

```
unique casez (a)
  2'b?0: ...
  2'b1?: ...
  default: ...
endcase
```

- Simulation and synthesis warn if "a" decodes to multiple branches (parallel_case)

```
unique if (a == 0) ...
else if (a == 2) ...
else if (a == 2) ...
else ...
```

© 2006

DVCon-2006: Proposed SystemVerilog Synthesis Subset

of 30

The Priority Decision Modifier



For more on unique and priority, download "SystemVerilog Saves the Day—the Evil Twins are Defeated! unique and priority are the new Heroes", at www.sutherland-hdl.com/papers

- The **priority** decision modifier...
 - Indicates that the decisions *must* be evaluated in order
 - Simulation has run-time warning if there is no matching condition
 - For synthesis, **priority** is equivalent to the pragma:
`//synthesis full_case`

given the declaration: `bit [1:0] a;`

```
priority case (a)
  0, 1: ...
  2   : ...
endcase
```

- Simulation warns if "a" has a value that is not decoded
- Synthesis ignores unspecified values (full_case)

```
priority if (a == 0) ...
else if (a == 2) ...
```

- **WARNING:** Synthesis compilers ignore the meaning of "priority"
 - Synthesis implements the full_case part of the priority modifier
 - Synthesis may optimize decision order without warning that the priority was changed — *at Sutherland HDL, we think this is wrong!*

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

22 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

Enhancements to Tasks and Functions



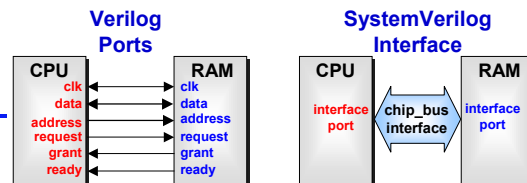
- SystemVerilog adds several synthesizable enhancements to Verilog tasks and functions
 - Default direction of input for formal arguments
 - Default data type of logic for formal arguments
 - Formal arguments can be any data type, including arrays, structures and user-defined types
 - Functions can have output and inout formal arguments
 - Void functions (called as a statement instead of an expression)
 - Function return values specified using return, as in C
 - Multiple statements implicitly grouped (don't need begin...end)
 - Task/function calls can use named argument passing
 - Passing arguments by reference (pointers)
 - Tasks and function can have a mix of static and automatic storage

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

23 of 30

Interfaces



- SystemVerilog interfaces are a compound port
 - Bundles any number of signals (nets and variables) together
 - Bundles port direction information with the signals
 - Bundles “methods” with the signals (e.g. a handshake sequence)
 - Bundles procedural functionality with the signals
 - Bundles assertion checks with the signals
- The synthesizable interface constructs are:
 - Interface definition ports
 - Interface data type declarations
 - Interface modport definitions
 - Interface methods (tasks and functions — must be automatic)
 - Interface procedural code

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

24 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

Declaring Module Ports As Interface Ports



- Modules ports can be declared as:
 - Generic interface** — any interface can be connected to the port
 - Explicit interface** — only the specified interface can be connected
- Beware of port mangling!** Synthesis may convert compound ports:
 - to a **single vector** that is a concatenation of the compound port
 - or**, to **separate ports** for each part of a compound port

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

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

```
module CPU (inout [103:0] pins_if);  
    ...  
endmodule
```

interface signals concatenated
into one large, bidirectional port

```
module CPU (input clk, output [31:0] address, inout [63:0] data, ...);  
    ...  
endmodule
```

interface signals expanded to separate ports

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

25 of 30

Module and Interface 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
```

can be verbose and redundant

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

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

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

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

26 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

What's Next



- ✓ Why we need a standard for synthesis
 - What major EDA vendors can synthesize
 - What problems this paper solves
- ✓ Synthesizable SystemVerilog Constructs
 - There's a lot!
- Recommendations
 - What EDA companies ought to do
- Summary



© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

27 of 30

Recommendations



- All constructs presented up to this point are currently supported by most synthesis compilers
 - Some synthesis companies are still implementing a few things
- Following are some constructs that are not currently supported, but which Sutherland HDL feels should be synthesizable
 - Task and function **ref** ports
 - Operator overloading
 - Bounded queues and queue methods
 - **case...inside** select statements
 - **case...matches** select statements
 - **uwire** single driver nets
 - Some synthesis compilers map **uwire** to **wire**, but that loses the single driver semantics — we recommend mapping **uwire** to **uwire**

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

28 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

Recommendation On Compound Port Mangling



- SystemVerilog allows ports to be a compound of several signals
 - Array ports, structure ports, union ports and interface ports
- Current synthesis compilers do not maintain compound ports in the synthesized results
 - Either create a large vector port by concatenating each component of the compound port together
 - Or, create separate ports for each component of the compound port
- A synthesized model cannot be directly plugged into a netlist that was expecting the compound ports
 - The net list must be modified to match the synthesized module ports
- Sutherland HDL recommends that synthesis compilers provide an option to create a wrapper module around synthesized modules
 - The wrapper module has compound ports like the original module, and maps those ports to an instance of the synthesized module

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset

29 of 30

Summary



- SystemVerilog adds hundreds of extensions to Verilog
 - Some extensions are intended to model hardware
 - Some extensions are intended for verification programs
- The IEEE has not defined a SystemVerilog Synthesis Subset
 - Synthesis companies must define their own subset
 - End users must create an even smaller subset that is supported by multiple vendors
- Sutherland HDL has worked with all major synthesis companies to define a common synthesis subset for SystemVerilog
 - We hope that the IEEE or Accellera will find this paper useful as a starting point for an official SystemVerilog Synthesis standard
 - We hope that design engineers will find this paper useful in lieu of a true IEEE SystemVerilog synthesis standard!

© 2006, Sutherland HDL, Inc.

DVCon-2006: Proposed SystemVerilog Synthesis Subset


30 of 30

A Proposal for a Standard SystemVerilog Synthesis Subset

DVCon-2006 Paper

by Sutherland HDL, Inc., Portland, Oregon

Questions?



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

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

© 2006, Sutherland HDL, Inc. DVCon-2006: Proposed SystemVerilog Synthesis Subset 31 of 30