Modeling with SystemVerilog in a Synopsys Synthesis Design Flow

Using Leda, VCS, Design Compiler and Formality

Stuart Sutherland
Sutherland HDL, Inc.
Portland, Oregon
stuart@sutherland-hdl.com

Training engineers to be HDL wizards

www.sutherland-hdl.com

What This Paper Will Cover

- What SystemVerilog offers design engineers
  - Overview of the SystemVerilog standard
  - Benefits of designing with SystemVerilog
- Synthesizable SystemVerilog Constructs
  - There are a lot!
- Some recommendations
  - Possible future enhancements of synthesis capabilities
- Summary

SystemVerilog is Several Languages Rolled Into One

- SystemVerilog combines the best of several design and verification languages
  - Verilog, VHDL, Superlog, VERA, C++, OVA, OVL, PSL, DirectC
- ***Advantages*** of merging the best of each language into one…
  - All members of design and verification teams use same language
  - Simulators use a single kernel — faster run time performance
  - Open standard — no proprietary, vendor-specific languages
- ***Disadvantages*** of a single design/verification language
  - More complex than Verilog or VHDL
  - Not easy to tell which parts of the language are for design (synthesizable)

This paper defines a SystemVerilog synthesis subset that works with Synopsys tools today!

SNUG EUROPE 2006

SystemVerilog is for Design Engineers, Too!

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

SystemVerilog

verification

- assertions
- test program blocks
- clocking domains
- process control
- mailboxes
- semaphores
- constrained random values
- direct C function calls

design

- interfaces
- nested hierarchy
- unrestricted ports
- automatic port connect
- enhanced literals
- time values and units
- specialized procedures
- packages
- 2-state modeling
- packed arrays
- array assignments
- queues
- unique/priority case/if
- compilation unit space

from C / C++

- classes
- inheritance
- strings
- dynamic arrays
- associative arrays
- references
- int
- shortint
- longint
- byte
- shortreal
- void
- alias
- globals
- enum
- typedef
- structures
- unions
- casting
- const
- break
- continue
- return
- do–while
- ++ -- += -= *= /=
- >>= <<= >>>= <<<=
- &= |= ^= %=

Verilog-2001

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

Verilog-1995

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

Benefits of SystemVerilog for Design Engineers

- ***SystemVerilog can substantially improve design productivity!***
  - Significantly reduce amount of code to represent complex designs
    - Code is easier to read and maintain
    - Fewer lines of code == fewer coding errors to debug
  - Dramatic reduction in mismatches between pre-synthesis RTL functionality versus post-synthesis gate-level functionality
  - White-box assertions without translate_off / translate_on pragmas
  - A common language is used throughout the design
  - Easy to integrate with SystemC models/testbenches

Design engineers can — AND SHOULD — be taking advantage of SystemVerilog *RIGHT NOW*!

Let's Look At The Details!

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

The $unit Shared Declaration Space

- SystemVerilog provides a built-in package called $unit
  - Any declaration outside of modeling blocks is in the $unit package
  - Automatically imported into all blocks compiled at the same time
- The synthesizable constructs that can be defined in $unit are:
  - Constant definitions
  - typedef user-defined types
  - Automatic task/function definitions

```
parameter MAX_SIZE = 128;
typedef enum {s1,s2,s3} states_t;
function automatic check_parity (...) ... endfunction

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

CAUTION! Each invocation of a compiler creates a unique $unit package; The paper contains recommendations on how to properly use $unit with VCS (multi-file compilation) and DC (single-file compilation)

User-defined Types and Enumerated Types

- SystemVerilog adds several forms of user-defined data types
  - The synthesizable user-defined types are **typedef** and **enum**
- **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 and explicit label values

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

- **typedef** defines a new type based on built-in types or other user-defined types (similar to C)

```
typedef  enum bit {FALSE, TRUE}  boolean_t;
```

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

Structures and Unions

- Structures collect multiple variables under a common name

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

  - Structure members 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};
```

- Unions define one storage space with multiple representations
  - Only packed unions are synthesizable

```
union packed {
  udp_t udp_packet;
  tcp_t tcp_packet;
} data_packet;
```

Type Casting, Size Casting 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
```

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!* DC converts compound ports to a single vector that is a concatenation of the compound port
  - The 2006.06 release will create a "wrapper module"

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!

Operators and Programming Statements

- SystemVerilog adds several synthesizable operators:
  - `++` and `--` increment and decrement
  - `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `^=`, `|=`, `<<=`, `>>=`, `<<<=`, `>>>=` assignment
- Synthesizable enhancements to Verilog programming statements:
  - Variables on left-hand side of continuous assignments
  - Enhanced **`for`** loops
  - **`foreach`** array traversal loop
  - **`break`**, **`continue`** and **`return`** jump statements
  - **`unique`** and **`priority`** decision statements (*important!*)
  - Simplified task and function definitions
  - **`void`** functions

Additional programming statement enhancements are presented in the paper

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 ...
```

The Priority Decision Modifier

- 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 **full_case** pragma:

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

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

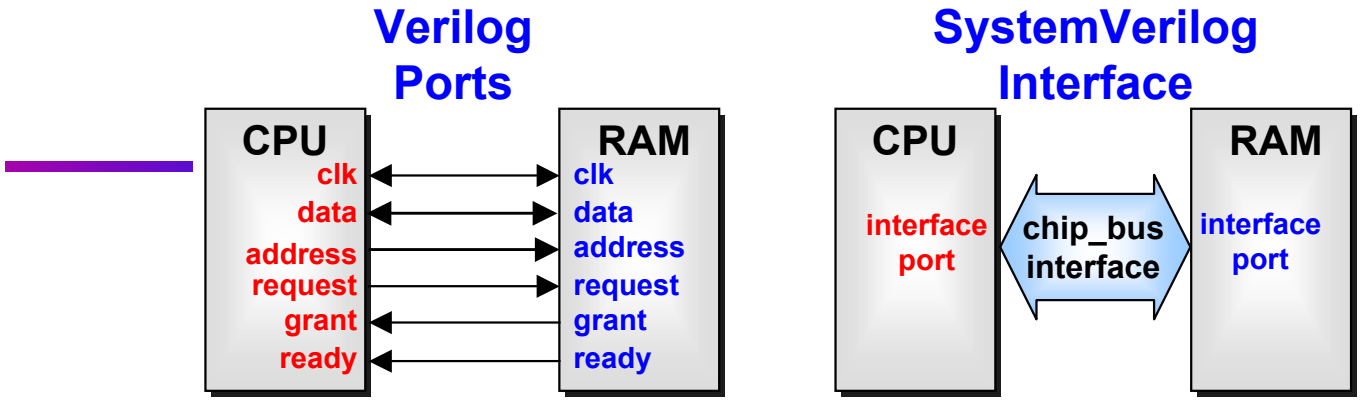
- Simulation warns if "a" has a value that is not decoded
- Synthesis ignores unspecified values (full_case)

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

- ***WARNING:*** DC might still optimize "priority" to parallel logic
  - If the case items are mutually exclusive, DC may optimize decision order without warning that the priority was changed

For more on unique and priority, download "SystemVerilog Saves the Day—the Evil Twins are Defeated!", presented at SNUG San Jose 2005

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

Declaring Module Ports As Interface Ports

- Modules ports can be declared as:
  - Generic interface — any interface can be connected to the port

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

  - Explicit interface — only the specified interface can be connected

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

- *Beware of port mangling!* DC converts interface ports to separate ports for each part of a compound port

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

interface signals expanded to separate ports

  - DC 2006.06 release will create a *"wrapper module"*
    - Has the original interface port(s)
    - Contains an instance of the synthesized module
    - Maps the interface ports to the separate ports

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);
  ...
endmod
```

```
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]));
```

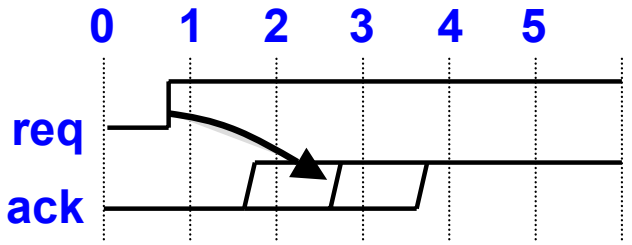
  - .* automatically connects all ports and nets with the same name

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

SystemVerilog Assertions

- SystemVerilog Assertions (SVA) add "white box verification" to complex design functionality
  - Concise syntax!

a complex sequence can be defined in very concise code




```
a_reqack: assert property (@(posedge clk) req ##[1:3] ack;) else $error;
```

  - Ignored by Synthesis!
    - Don't have to hide checker code within convoluted translate_off / translate_on synthesis pragmas
  - Can be disabled!
    - Can be turned off during reset design or simulation conditions

For more on assertions in designs, see "SystemVerilog Assertions are for Design Engineers, too!", presented at SNUG San Jose 2006 and SNUG Europe 2006

Recommendations

- The constructs presented are supported by Synopsys tools
  - A couple of constructs are still in beta test (see paper for details)

Designers should use SystemVerilog NOW! The constructs supported by Synopsys enable design engineers to be more productive today

- There are some constructs that are not currently supported, but which may be synthesizable in the future
  - Packages
  - User-defined net data types
  - `==?` and `!=?` wild equality/inequality operators (similar to `casex`)
  - Operator overloading
  - `case`…`inside` select statements
  - `case`…`matches` select statements
  - `uwire` single driver nets

Let your Synopsys rep know what additional constructs are important in your design projects!

Summary

- SystemVerilog adds hundreds of extensions to Verilog
  - Some extensions are intended to model hardware
  - Some extensions are intended for verification programs
- The SystemVerilog design constructs are important!
  - Enable design engineers to design more hardware in less time
  - Significantly reduces the risk of design errors
- Synopsys has implemented a SystemVerilog synthesis subset
  - Supported by all Synopsys tools used in a synthesis design flow
    - Leda, VCS, Design Compiler, Formality
- This paper identifies the Synopsys SystemVerilog synthesis subset currently supported in a full Synopsys design flow
  - We hope that design engineers will find this paper useful as they adopt SystemVerilog in current and future projects!