Can My Synthesis Compiler Do That?

Important SystemVerilog Features Supported by ASIC and FPGA Synthesis Compilers

Stu Sutherland
Sutherland HDL

Don Mills
Microchip Technology

March 4, 2014

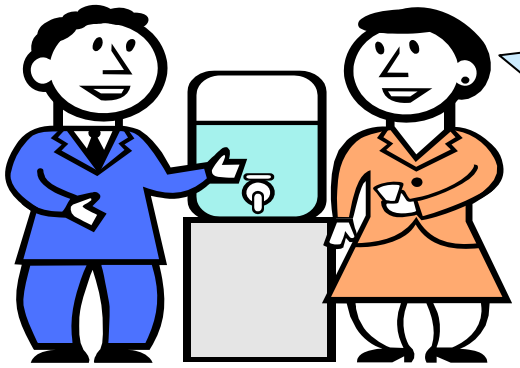
What This Paper is About…

- ✓ Debunking the myth that SystemVerilog is not synthesizable
- ✓ Several important SystemVerilog constructs that are synthesizable
- ✓ Why those constructs are important for you to use
- ✓ 18 major advantages designing with 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

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
  - ASIC and FPGA synthesis compilers have excellent support for RTL modeling with SystemVerilog

SystemVerilog is Synthesizable

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	

Can My Synthesis Compiler… Tell Me Where My Design Logic Type is Functionally Incorrect?

- Verilog always procedures model all types of design logic
  - Synthesis must "*infer*" (guess) whether an engineer intended to have combinational, latched or sequential functionality

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

where did all these latches come from?

- SystemVerilog has hardware-specific always procedures: **`always_comb`**, **`always_latch`**, **`always_ff`**
  - Documents designer intent
  - Enforces several synthesis RTL rules
  - Synthesis can check against designer intent

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

```
Warning: test.sv:5:
Netlist for always_comb
block contains a latch
```

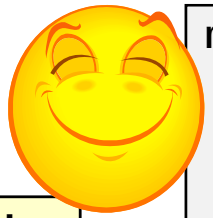
- **What's the advantage?**
  - ✓ Non-synthesizable code gives warnings
  - ✓ Simulation, synthesis and formal tools use same rules

Can My Synthesis Compiler… Automatically Infer wire and reg Data Types?

- Traditional Verilog has strict and confusing rules for port types
  - Input ports must be a net type (**`wire`**)
  - Output ports must be:
    - **`reg`** in some contexts
    - **`wire`** in other contexts

```
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?**
  - ✓ Defining and maintaining modules just got a whole lot easier!

Can My Synthesis Compiler… Allow Me to Avoid Using Cryptic Vector Part Selects?

- A byte (or other segment size) of a traditional Verilog vector is accessed using part-selects
  - Awkward to use if a design frequently works with bytes
  - Part-select *coding errors will often simulate and synthesize*
- SystemVerilog vectors can be declared with subfields
  - Subfields are accessed using a simple index instead of part-selects

```
module add_third_byte
 (input  wire [31:0] a, b,
  input  wire        ci,
  output reg   [7:0] sum,
  output reg         co );
  always @(a, b, ci)
   {co,sum} = a[24:17] + b[24:17];
endmodule
```

```
module add_third_byte
 (input  logic [2:0][7:0] a, b,
  ... );
  always_comb
   {co,sum} = a[3] + b[3];
```

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

- **What's the advantage?**
  - ✓ Byte selects are easier to model and are correct by construction!

Can My Synthesis Compiler…

Prevent Me From Stupid Mistakes?

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
```

6 functional bugs (must detect, debug and fix)

SystemVerilog adds enumerated types

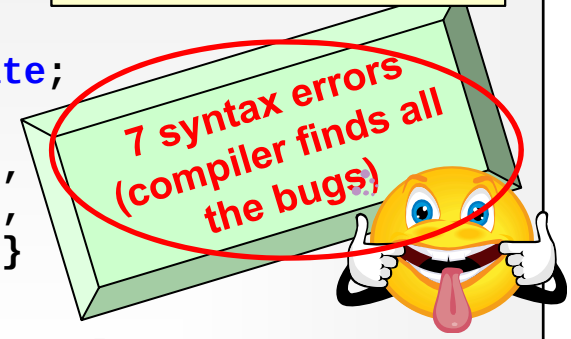
```
enum logic [2:0]
  {WAIT = 3'b001,
   LOAD = 3'b010,
   DONE = 3'b001}
  state, next_state;
enum logic [1:0]
  {READY = 3'b101,
   SET   = 3'b010,
   GO    = 3'b110}
  mode_control;

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
```

7 syntax errors (compiler finds all the bugs)



Can My Synthesis Compiler… Allow Me to Transfer a Look-Up Table in One Line of Code?

- In Verilog arrays could only be accessed one element at a time
  - Hard to reset, load or copy
  - Hard to pass through module ports or to a function

```
reg [7:0] t1[0:63], t2[0:63];
integer i;
always @(posedge clk)
  if (load)
    for (i=0; i<=63; i=i+1)
      t2[i] <= t1[i];
```

- SystemVerilog allow entire arrays to be assigned
  - Load array with one statement
  - Copy array with one statement
  - Pass through ports or to a function

```
always @(posedge clk)
  if (!rstN) t2 <= '{default:0};
  else if (load) t2 <= t1;
```

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

Can My Synthesis Compiler… Allow Me to Reduce 216 Lines of Code to Just 4 Lines?

- Traditional Verilog has no easy way to bundle related signals
  - Each signal is treated separately for assigning, copying, passing through module ports, etc.
- SystemVerilog structures bundle related signals together
  - Entire structure can be assigned, copied, or passed through module ports

```
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 [0:47];
} uni_t; // UNI cell bundle
```

a UNI ATM cell is made up of 54 signals

```
module cell_transmitter
 (output uni_t  d_out,
  input  uni_t  d_in,
  input  logic  clk, rstN);

  always @(posedge clk or negedge rstN)
    if (!resetN) d_out <= '{default:0};
    else         d_out <= d_in;
endmodule
```

54 ports in Verilog

another 54 ports

108 separate assignment statements in Verilog

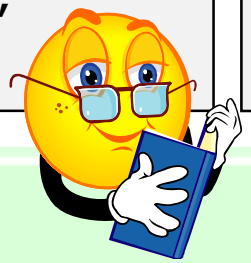
- **What's the advantage?**
  - ✓ Structures reduce code and ensure consistency

Can My Synthesis Compiler… Support Sharing Definitions Across Several Modules?

- 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

Can My Synthesis Compiler…

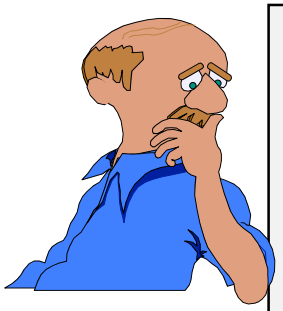
Help Me Make Better Decisions?

- Verilog has a limited number of decision statements
  - Comparing to multiple values can require extra code
  - The **`casex`** and **`casez`** statements have major "gotchas"
- SystemVerilog adds:
  - **`inside`** set membership operator

```
if (data inside {[0:255]) ...
```

if data is between 0 to 255, inclusive

  - **`case() inside`** wildcard decisions



```
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?**
  - ✓ Decisions can be modeled more accurately and with less code

Can My Synthesis Compiler… Reverse All the Bits or Bytes of My Resizable Vector?

- Verilog requires that bit-selects and part-selects of vectors must be referenced using the same endian as the vector declaration
  - Makes it difficult to swap the order of bits or bytes of a vector

```
parameter N=64;
reg [N-1:0] d;
```

bit reverse

```
always @(posedge clk)
  if (swap_bits)
    for (i=0; i<N; i=i+1)
      d[(N-1)-i] <= d[i];
```

byte reverse

```
always @(posedge clk)
  if (swap_bytes)
    for (i=0; i<N; i=i+8)
      d[((N-1)-i)-:8] <= d[i+:8];
```

Huh?

- SystemVerilog adds pack and unpack streaming operators

SystemVerilog bit reverse

```
d = { << { d }};
```

SystemVerilog byte reverse

```
d = { <<8{ d }};
```

- **What's the advantage?**
  - ✓ Data manipulation is easier to code and less likely to have errors

Can My Synthesis Compiler… Eliminate False Warnings for Intentional Size Mismatches?

- Verilog does automatic size and type conversions
  - RTL code often depends on these conversions
  - Can result in false warning messages from lint checkers

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

The truncation will cause a synthesis warning, even though the code is correct

Rotate a by b number of times – depends on upper 32 bits of the 64-bit operation result being truncated

- SystemVerilog has a cast operator
  - Explicit conversion do not cause warnings

```
y = 32'({a,a} >> b);
```

cast the operation result to 32 bits before assigning

- **What's the advantage?**
  - ✓ Documents intent that a change in type, size or sign is intended
  - ✓ Can eliminate size and type mismatch warnings

Can My Synthesis Compiler… Prevent Subroutines that Simulate but won't Synthesize?

- Verilog tasks are subroutines
  - Synthesis imposes several restrictions on tasks
  - It is possible for a task to simulates correctly, but not synthesize
  - Verilog functions will almost always synthesize correctly, but cannot be used in place of a task
- SystemVerilog enhances functions several ways
  - Void functions – functions that can be used like a task
  - Functions with output and inout formal arguments
  - Passing arrays and structures as arguments (and more)

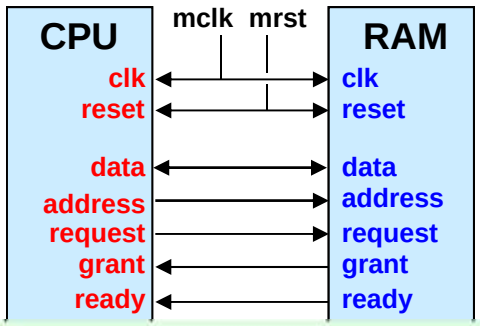
Important for synthesis!

- **What's the advantage?**
  - ✓ Void functions can be used like tasks, but are still functions and help ensure that the subroutine will be synthesizable

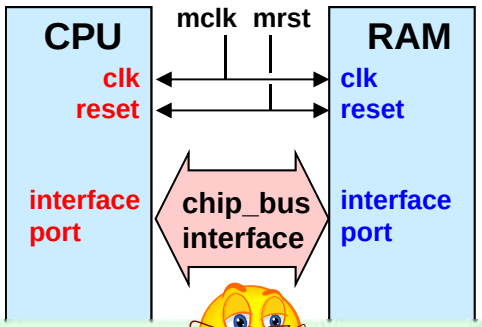
Can My Synthesis Compiler… Support Modeling Bus Protocols at a High Level of Abstraction?

- Verilog uses separate ports for each signal in a bus protocol
  - Requires lots of redundant port declarations and complex netlists
- SystemVerilog adds interfaces – compound, multi-signal ports
  - Bundles bus protocol signals together – can include subroutines

Verilog discrete ports



SystemVerilog interface ports




```
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

Can My Synthesis Compiler… Easily Fill Expression Vectors of Any Size with All 1s

- 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

Can My Synthesis Compiler… Automatically Calculate the Size of My Address Busses?

- Verilog does not have a built-in way to calculate vector widths
  - Engineers must calculate vector widths
  - A mistake, or a change in design spec, can result in a faulty design
- SystemVerilog adds:
  - **$bits** returns the width of a vector
  - **$clog2** returns the ceiling log base 2 of a vector width

```
module fifo
#(parameter FIFO_SIZE = 32)
 (input  wire       i_clk, o_clk,
  input  wire [4:0] rd_ptr, wr_ptr,
  input  wire [7:0] data_in,
  output reg  [7:0] data_out
);
  ...
```

What if the FIFO SIZE changes?

```
module fifo
#(parameter FIFO_SIZE = 32,
  P_WIDTH = $clog2(FIFO_SIZE)
 (input  logic i_clk, o_clk,
  input  logic [P_WIDTH-1:0] rd_ptr,
                             wr_ptr,
  ... );
  ...
```

The pointer widths automatically scale to the FIFO SIZE

- **What's the advantage?**
  - ✓ Vector sizes are scalable and correct by construction!


These are great additions to traditional Verilog!
Yes, but does our synthesis compiler support SystemVerilog?

SystemVerilog Synthesis Support

- GREEN = supported
- YELLOW = partially supported
- RED = not supported

SystemVerilog Construct	Compiler A	Compiler B	Compiler C	Compiler D	Compiler E	Compiler F
always_ff, always_comb, always_latch	✓	✓	✓	✓	✓	✓
logic type / inferred wire or reg	✓	✓	✓	✓	✓	✓
Vectors with subfields	✓	✓	✓	✓	✓	✓
Enumerated types	✓	✓	✓	✓	✓	✓
Array assignments and copying	✓	✗	✓	✓	✓	✓
Structures and structure assignments	✓	✓	✓	✓	✓	✓
Packages and package import	✓	✓	✓	P	✓	✓
inside and case…inside decisions	✓	✓	✓	P	✓	✗
Streaming operators	✓	✓	✓	✓	✓	✗
Casting	✓	✓	✓	✓	✓	✓
Void functions	✓	✓	✓	✓	✓	✓
Interfaces	✓	✓	✓	✓	✓	✓
Vector fill tokens	✓	✓	✓	✓	✓	✓
$bits, $clog2 expression size function	✓	✓	✓	✓	✓	✓

See the paper for details

But Wait… There's More

- SystemVerilog has many more *useful constructs* that are supported by many synthesis compilers
  - Version keyword compatibility
  - 2-state types
  - User-defined types
  - Parameterized types
  - Unions
  - ++ and -- increment/decrement
  - unique/priority decision checkers
  - Multiple for-loop iterator variables
  - do...while loops
  - foreach loops
  - break and continue loop controls
  - Continuous assignments to variables
  - Task/function formal arguments with default values
  - Task/function calls pass by name
  - Function return statements
  - Parameterized tasks and functions
  - Dot-name and dot-star netlist connections
  - Named statement group ends
  - const variables
  - Assertions
  - Local time unit and precision
  - $unit declaration space

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
- ASIC and FPGA synthesis compilers support SystemVerilog
  - There are some differences, but overall support is very good
- 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

Once Upon a Time…

Four design engineers worked on an important design.

There names were: **Somebody**, **Everybody**, **Anybody** and **Nobody**.

Everybody had attended this DVCon paper on synthesizing SystemVerilog, and was sure that **Somebody** would take advantage of using SystemVerilog. **Anybody** could have written the RTL code in SystemVerilog, but **Nobody** did it.

Instead, the design was modeled in Verilog-2001 RTL using redundant, error-prone code. The product missed its market window, and cost the company oodles of lost revenue.

Everybody blamed **Somebody** because **Nobody** did what **Anybody** could have done (but their competitors were pleased).