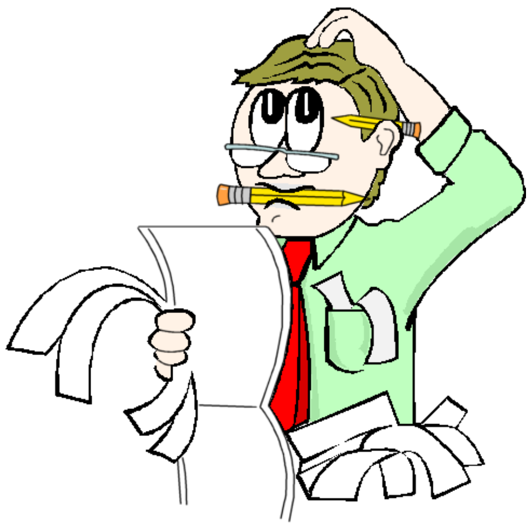
SNUG SAN JOSE 2007



More Standard Gotchas
Subtleties in the Verilog and SystemVerilog Standards That Every Engineer Should Know!

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

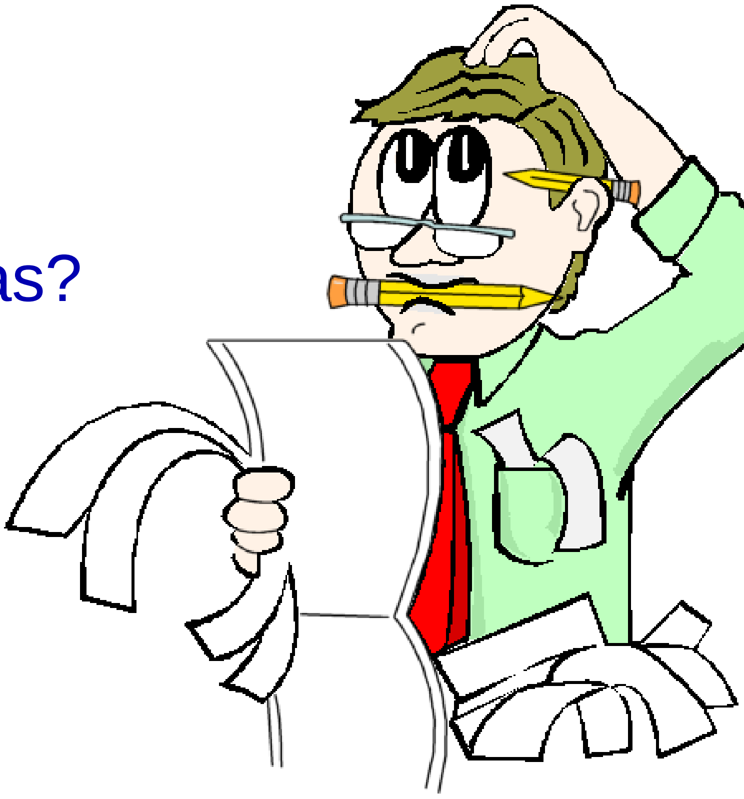
Don Mills
LCDM Engineering
Chandler, Arizona
mills@lcdm-eng.com

Chris Spear
Synopsys, Inc.
Marlboro, Massachusetts
chris@spear.net

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

Presentation Overview

- What is a “gotcha”?
- Why do standards have gotchas?
- What’s covered in this paper
- Several example gotchas,
  and how to avoid them!
- Summary



What Is A Gotcha?

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- In programming, a "gotcha" is a legal language construct that does not do what the designer expects

A Classic C programming Gotcha...

```
if (day = 15)
  /* process payroll */
```

Gotcha!

~~If middle of the month, then pay employees…~~

GOTCHA! This code will assign the value of 15 to day, and then if day is not zero, pay the employees

```
if (day == 15) ...
```

- In hardware design and verification, most gotchas will simulate, but give undesired results
  - Gotchas can be difficult to find and debug
  - A gotcha can be disastrous if not found before tape-out!

Engineers need to know how to recognize and avoid gotchas in hardware modeling!

Why Do Standards Have Gotchas?

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- ~~Standards developers are idiots~~
- ~~Users of standards are idiots~~
- Languages can be used the right way, or the wrong way

```
if (day = 15)
  /* process payroll */
```

Gotcha!

A dumb way to use "assignment within an expression"

```
while (data = fscanf(…))
  /* read in data until it is 0 */
```

A clever way to use "assignment within an expression"

- Hardware models are not just simulated, they are synthesized, analyzed, emulated, prototyped, formally proved, …
  - Each type of tool needs different information from the language
- Verilog and SystemVerilog allow designers to prove what will — and what will not — work correctly
  - Models that won't work correctly need to be legal syntax

Is This a Verilog Gotcha?

Stu Sutherland
Sutherland HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- Is the classic C gotcha also a gotcha in Verilog?

```
always @(state)
  if (state = LOAD)
    ...
```

Legal or Illegal?

Illegal! Verilog does not allow assignment statements inside of expressions

- What about in SystemVerilog?
  - SystemVerilog extends Verilog with more C and C++ features

```
always @(state)
  if (state = LOAD)
    ...
```

Legal or Illegal?

If you don't know the answer, then you really need to read this paper!

(We will answer this question at the end of our presentation...)

Standard Gotcha's, Part One (SNUG-Boston 2006)

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- Detailed descriptions of 57 gotchas…and how to avoid them!

- Case sensitivity
- Implicit net declarations
- Escaped identifiers in hierarchy paths
- Verification of dynamic data
- Variables declared in unnamed blocks
- Hierarchical references to package items
- Variables not dumped to VCD files
- Shared variables in modules
- Shared variables in interfaces, packages
- Shared variables in tasks and functions
- Importing enum types from packages
- Importing from multiple packages
- Resetting 2-state models
- Locked state machines
- Hidden design problems
- Out-of-bounds indication lost
- Signed versus unsigned literal integers
- Default base of literal integers
- Size mismatch in literal integers
- Literal size mismatch in assignments
- Z extension backward compatibility
- Filling vectors
- Passing real types through ports
- Port connection rules
- Back-driven input ports
- Self- & context-determined operations
- Operation size and sign extension
- Signed math operations
- Bit and part select operations
- Increment and decrement operations
- Pre-increment versus post-increment
- Multiple read/writes in one statement
- Operator evaluation short circuiting
- Assignments in expressions
- Procedural block activation
- Combinational logic sensitivity lists
- Arrays in sensitivity lists
- Vectors in sensitivity lists
- Operations in sensitivity lists
- Sequential blocks with begin...end
- Sequential blocks with partial reset
- Blocking assigns in sequential blocks
- Evaluation of true/false on 4-state values
- Not operator versus invert operator
- Nested if...else blocks
- Casez/casex masks in case expressions
- Incomplete or redundant decisions
- Out-of-bounds in enumerated types
- Statements that hide design problems
- Simulation versus synthesis mismatches
- Multiple levels of same virtual method
- Event trigger race conditions
- Using semaphores for synchronization
- Using mailboxes for synchronization
- Coverage reporting
- $unit declarations
- Compiling $unit

Standard Gotcha's, Part Two (SNUG-San Jose 2007)

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- 38 additional gotchas…and how to avoid them!

- Overlapped decision statements
- Full_case or unique case synthesis gotcha
- Combinational logic hidden storage
- Nonblocking assignments in comb. logic
- Memory models that can't be loaded
- Default of 1-bit internal nets
- Port direction coercion
- Compile errors with clocking blocks
- Misplaced semicolons on begin...end
- Misplaced semicolons in if...else
- Misplaced semicolons in loops
- Unintentional infintite loops
- Locked simulation with concurrent loops
- Using loop variables outside the loop
- Summing a subset of an array
- Task/function arguments with defaults
- Static task/function gotcha
- Local variables error
- Program statements in classes
- Using interfaces with classes
- Mailboxes that forget stored values
- Passing handles to functions
- Creating arrays of objects
- Some variables won't randomize
- Boolean randomization constraints that do weird things
- Undectected randomization failures
- Unwanted negative random values
- Coverage is always 0%
- Coverage report results get lumped together
- Covergroup directions that are not what they seem
- Assertion pass statements execute at wrong time
- Assertion fail statements execute at wrong time
- Procedural assertion gotcha
- Default time units
- Package chaining
- Non-standard keywords
- Array literals versus concatenations
- Declaring floating point ports

(The highlighted gotchas are the ones we will discuss in this presentation)

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

A Synthesis Gotcha: Defaults with Full_case or Unique Case

- Default assignments before a case statement simplifies code
- Using full_case or unique case can avoid unintended latches
  - ***But…*** *combining the two styles is a gotcha!*

```
always_comb begin
  load_s    = 1'b0;
  load_f    = 1'b0;
  load_pc   = 1'b0;
  inc_pc    = 1'b0;
  set_br    = 1'b0;
  dmem_rdN  = 1'b0;
  dmem_wrN  = 1'b0;
  case (state) // synopsys full_case
    state1 : inc_pc = 1'b1;
    state2 : set_br = 1'b1;
    ...
  endcase
end
```

default values for combinational outputs (could be several dozen outputs)

Gotcha!

DC treats a full_case or unique case statement as fully self-contained (any assignments to the same variables before the case statement are ignored!)

only values different than default are listed in case statement (could be dozens of branches)

- ***To avoid this Gotcha…*** Don't mix default assignments with **`full_case`** or **`unique case`** decisions

Gotcha: Unintentional Infinite Loops

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- For-loops typically exit when a control variable exceeds some limit
  - **But…***it is possible to declare variables that cannot hold the loop exit value*

```
integer sb[0:15];
reg [3:0] i;
initial begin
  ... // do lots of tests...
  for (i=0; i<=15; i=i+1) begin
    $display("sb[%0d]=%0d", i, sb[i]);
  end
  $finish;
end
```

Gotcha!

- ***To avoid this Gotcha…***
  - Use the **`int`** type for for-loop control variable
  - Use SystemVerilog style for-loops

```
for (int i=0; i<=15; i=i+1)
```

    - Makes the loop variable type more obvious

Gotcha: Using Loop Variables Outside of the Loop

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- For-loop variables can be declared two different ways
  - Outside of the loop (Verilog style)
  - As part of the loop (SystemVerilog style)
    - ***But…****it is a local variable that cannot be used outside the loop!*

Verilog Style for Loop Variable

```
integer a[0:31], b[0:31];
int i;
initial begin
  for (i=0; i<=31; i=i+1) begin
    if (a[i] != b[i]) break;
  end
  if (i < 32)
    $display("Mismatch at %0d", i);
end
```

SystemVerilog Style for Loop Variable

```
integer a[0:31], b[0:31];
initial begin
  for (int i=0; i<=31; i++) begin
    if (a[i] != b[i]) break;
  end
  if (i < 32)
    $display("Mismatch at %0d", i);
end
```

Gotcha!

Variables declared as part of the for-loop are local to just the loop

- ***To avoid this Gotcha…***
  - When the loop control variable needs to be used outside of the loop, use the Verilog style of loop control variable

Gotcha: Using Programming Statements in Classes

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- Classes enable writing Object Oriented re-usable testbenches
  - ***But…**classes cannot directly execute programming statements*
    - Classes contain "methods" that are called from procedural code

Definition of an object

```
class Foo;
  int data;
  function int get (...);
    ...
  endfunction
  task put (...);
    ...
  endtask
endclass
```

Definition of another object

```
class Bar;
  Foo f = new;
  f.data = 3;
endclass
```

Instantiate object Foo and initialize it's data

Gotcha!

A class cannot directly execute programming statements

- ***To avoid this Gotcha…***
  - See the next slide

Using Programming Statements in Classes (continued)

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- Classes enable writing Object Oriented re-usable testbenches
  - ***But…****classes cannot directly execute programming statements*
- ***To avoid this Gotcha…***
  - Use "methods" to hold class programming statements
  - The **new** method can be used to initialize class variables

Definition of an object

```
class Foo;
  int data;
  function new (int d);
    this.data = d;
  endfunction
  function int get (...);
    ...
  endfunction
  task put (...);
    ...
  endtask
endclass
```

The assignment statement is in a method

Definition of another object

```
class Bar;
  Foo f = new(3);
  ~~f.data = 3;~~
endclass
```

Initialize Foo's data using its new method instead of executing a programming statement!

Gotcha: Randomization Constraints That Don't Work

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- Constraints are used to set limits on randomly generated values
  - **But…***multiple Boolean constraints may limit values differently than intended*

Intent: Constrain values such that lo is less than med, and med is less than hi

```
class bad1;
  rand bit [7:0] lo, med, hi;
  constraint increasing { lo < med < hi; }
endclass
```

Gotcha!

Sample of values generated:

```
lo =  20, med = 224, hi = 164
lo = 114, med =  39, hi = 189
lo = 186, med = 148, hi = 161
```

Why is med greater than hi?

Why is lo greater than med?

Constrains hi to be greater than result of true/false test of "lo < med"
- lo and med are not constrained
- Result of true/false test is 0 or 1

- ***To avoid this Gotcha…***
  - Constraints involving multiple Boolean operations should be broken into multiple statements

```
constraint increasing {
    lo < med;
    med < hi; }
```

Gotcha: Assertion Pass Statements Execute at the Wrong Time

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- SystemVerilog Assertions can execute an optional "pass statement" whenever the assertion succeeds

[*name* :] **assert property (** *property_specification* **)** [*pass_statement*] [**else** *fail_statement*] **;**

```
property p_req_ack;
  @(posedge clk) req |-> ##1 ack;
endproperty

assert property (p_req_ack) req_ack_count++;
```

- If req then check for ack on next clock cycle
- If no req then abort with "vacuous success"

Intent: Count how many times req is followed by ack

Gotcha!

Counts how many times req is followed by ack *and* how many times there was no req

  - ***But…****pass statements execute on both Success and Vacuous Success*

- ***To avoid this Gotcha…***
  - The next version of SystemVerilog will have a control to only execute pass statements on success

Gotcha: Assertion Fail Statements Execute at the Wrong Time

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

- SystemVerilog Assertions can execute an optional "fail statement" whenever the assertion fails

[*name* :] **assert property (** *property_specification* **)** [*pass_statement*] [**else** *fail_statement*] **;**

```
property p_req_ack;
  @(posedge clk) req |-> ##1 ack;
endproperty

assert property (p_req_ack)
  if (cnt_en) req_ack_count++;
else $fatal;
```

Pass statement

Fail statement?

Gotcha!

Intent:
- **Increment counter if assertion passes and `cnt_en` is true**
- **Exit with fatal error if assertion fails**

The `else` is paired with the `if`, instead of the `assert`

  - ***But...*** *the assertion* **`else`** *can be inadvertantly paired with an* **`if`** *statement that is part of the pass statement*
- ***To avoid this Gotcha…***
  - Add begin...end around the pass if statement

SNUG SAN JOSE 2007

Gotcha: Assignments in Expressions

Stu Sutherland
SUTHERLAND HDL

Don Mills
LCDM Engineering

Chris Spear
Synopsys

- Is the classic C gotcha also a gotcha in SystemVerilog?

```
always @(state)
  if (state = LOAD)
    ...
```

Legal or Illegal?

- SystemVerilog allows assignments in expressions…
  - **But,…** *the syntax is different than C — the assign statement must be enclosed in parentheses*

```
always @(state)
  if ( (state = LOAD) )
    ...
```

Gotcha!

The different syntax helps prevent the gotcha of using = where == is intended, but…
The different syntax is confusing to C/C++ programmers when an assignment is intended

- ***To avoid this Gotcha…***
  - "It is what it is" — Engineers need to learn the unique SystemVerilog syntax

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

Summary

- Programming languages have "gotchas"
  - A legal construct used in a way that gives unexpected results
  - Gotchas occur because useful language features can be abused
- A gotcha in a hardware model can be disastrous
  - Difficult to find and debug
  - If not found before tape-out, can be very costly
- This paper describes 38 Verilog and SystemVerilog gotchas
  - Detailed explanations of each gotcha
  - Guidelines on how to avoid each gotcha
  - Lots of code examples
- This paper is Part 2
  - Part 1 was presented at SNUG-Boston 2006

Questions & Answers...

Stu Sutherland
SUTHERLAND HDL
Don Mills
LCDM Engineering
Chris Spear
Synopsys

Do you have a favorite gotcha that is not in the Part 1 or Part 2 paper?

Please send it to Stu or Don!

We are collecting gotchas for publication in a book...

stuart@sutherland.com

mills@lcdm-eng.com