SNUG SAN JOSE 2005

# SystemVerilog Saves the Day, the Evil Twins are Defeated

## "unique" and "priority" are the new heroes!

Stuart Sutherland

Sutherland HDL, Inc.

Portland, Oregon

stuart@sutherland-hdl.com

SNUG SAN JOSE 2005

# Objectives

SUTHERLAND HDL
Training Engineers to be HDL wizards

- This paper is on the proper modeling, simulation and synthesis of multiple-branch decisions

  - Review how Verilog decision statements simulate
  - Review how Verilog decision statements synthesize
  - Discuss the full_case and parallel_case synthesis pragmas, and why they are evil villains!
  - Discuss the SystemVerilog **`unique`** and **`priority`** decision modifiers, and why they are heroes!
- The goals of this paper (and every engineer) are:
  - To ensure that the design engineer's assumptions about multiple-branch decisions are correct
  - To ensure that synthesis will correctly implement what the design engineer specified

# The Verilog if…else Decision Statement

SUTHERLAND HDL
Training Engineers to be HDL wizards

SNUG SAN JOSE 2005

- The `if... else` statement controls whether a statement, or group of statements, should be executed
  - Executes the first branch if expression is true
  - Executes the second branch if the expression is false or unknown
  - Can nest `if` decisions within `else` branches for multiple decisions
    - Only the first matching branch will be executed
    - At most, only 1 branch will be executed!

These rules affect synthesis!

  - The `else` branch is optional
    - If no `else` branch, and the `if` condition is false or unknown, then no branch is executed

This rule affects synthesis!

```
module mux3to1 (output reg  y,
                input       a, b, c,
                input [1:0] select );
  always @* begin
    if      (select == 2'b00) y = a;
    else if (select == 2'b01) y = b;
    else if (select == 2'b10) y = c;
  end
endmodule
```

**In simulation, if select is 2'b11, y retains its previous value!**

# The Verilog case Decision Statement



- The `case` statement compares a *"case expression"* to multiple *"case item expressions"*
  - Executes the branch for the first matching case item
  - At most, only 1 branch will be executed!
  - Executes a `default` branch if no match is found
    - The default branch is optional
    - If no default branch, and the case expression does not match any case item expressions, then no branch is executed

These rules affect synthesis!

This rule affects synthesis!

```
module mux3to1 (output reg  y,
                input       a, b, c,
                input [1:0] select );
  always @* begin
    case (select)
      2'b00:    y = a;
      2'b01:    y = b;
      2'b10:    y = c;
      default: y = 'bx;
    endcase
  end
endmodule
```

"case expression"

"case item expressions"

**what would happen if the optional default branch was left off, and select was 2'b11?**

# The casez and casex Decision Statements

SUTHERLAND HDL
Training Engineers to be HDL wizards

SNUG SAN JOSE 2005

- Special versions of the case statement allow the X and Z logic values to be used as "don't care" bits in the comparison:
  - **casez** uses **Z** values as don't care instead of as a logic value (a **?** in a literal number is treated as a **Z**)
  - **casex** uses either **X** or **Z** values as don't care instead of as logic values (a **?** in a literal number is treated as a **Z**)

```
casez (opcode)
  4'b1zzz: out = a;      // don't care about lower 3 bits
  4'b01??: out = b;      // don't care about lower 2 bits
  4'b001?: out = a + b; // don't care about lower bit
  default: out = 32'bx;
endcase
```

**For guidelines on the proper use of casez and casex, see *"RTL Coding Styles That Yield Simulation and Synthesis Mismatches"*, Mills and Cummings, SNUG San Jose, 1999**

# Software Behavior versus Hardware Behavior

SNUG SAN JOSE 2005

SUTHERLAND HDL
*Training Engineers to be HDL wizards*

- Software simulation can only approximate hardware behavior
  - Software programs execute sequentially, one instruction at a time
  - Hardware can evaluate logic in parallel

```
module mux3to1 (output reg  y,
                input       a, b, c,
                input [1:0] select );
  always @* begin
    if      (select == 2'b00) y = a;
    else if (select == 2'b01) y = b;
    else if (select == 2'b10) y = c;
  end
endmodule
```

**Software (e.g. RTL simulation)**

- **Will evaluate the select values in the order in which they are listed**
- **If select == 2'b11, no branch is executed (y remains unchanged)**

**Hardware (e.g. logic gates)**

- **Can evaluate all select values in parallel, with no specific order**
- **If select == 2'b11, something will happen to y, but what?**

**For hardware to match software, latches are required to retain the previous value of y**

# Software versus Hardware: Interrupt Decode Example

SUTHERLAND HDL
*Training Engineers to be HDL wizards*

SNUG SAN JOSE 2005

- Extra logic is required to make hardware behave like software

```
module interrupt_decode (output reg  something,
                         input [3:0] IRQ );
  always @* begin
    if      (IRQ[0])    // process interrupt 0
    else if (IRQ[1])    // process interrupt 1
    else if (IRQ[2])    // process interrupt 2
    else if (IRQ[3])    // process interrupt 3
    else                // process default for no interrupt
  end
endmodule
```

**Software (e.g. RTL simulation)**

- **Will evaluate the IRQ bits in the order in which they are listed (bit 0 has highest priority)**
- **If two or IRQ bits are set, only the branch for the lowest number bit is executed**

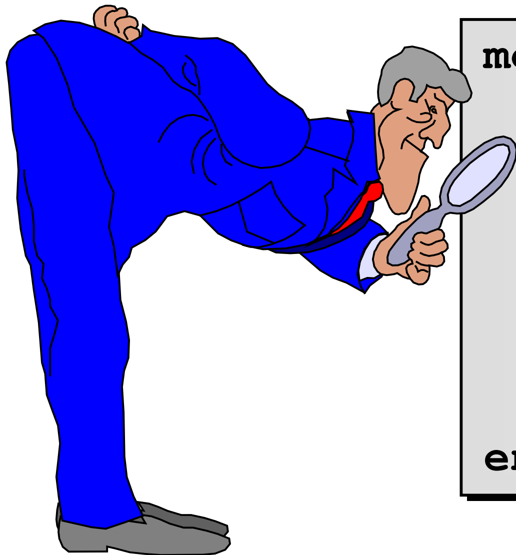
**In order for hardware (e.g. logic gates) to match software behavior**

- **Priority encoding must be added to the logic**
- **Adds more gates and longer timing paths to the design**

# Synthesizing Decision Statements



- The DC synthesis tool looks for two key factors when translating Verilog decision statements into hardware



```
module mux3to1 (output reg   y,
                input        a, b, c,
                input  [1:0] select );
  always @* begin
    if      (select == 2'b00) y = a;
    else if (select == 2'b01) y = b;
    else if (select == 2'b10) y = c;
  end
endmodule
```

- Is the decision statement fully specified?
- Can the decision expressions be evaluated in parallel?

**DC only considers 2-state values when evaluating a decision statement**
- **Logic values X and Z are not considered**

# Fully Specified Decisions



- A fully specified decision ensures that all variables are assigned a value for all possible 2-state values of the controlling expression
- For `if...else` decisions:
  - Either every `if` branch must have a matching `else` branch,
  - Or, all variables must be assigned a value prior to the `if` decision
- For `case`/`casez`/`casex` statements:
  - Either all possible values of the case expression must have a matching case item expression
  - Or, there must be a default assignment to for any unspecified case expression values
    - Can be a `default` branch within the case statement
    - Can be a default assignment prior to the case statement

**There are additional requirements for fully specified decision statements that are covered in the paper, but not in this presentation**

# Example of a Fully Specified Decision

SNUG San Jose 2005

SUTHERLAND HDL
*Training Engineers to be HDL wizards*

- A fully specified decision statement defines a branch of execution for all possible 2-state values of the controlling expression

```
module mux4to1 (output reg   y,
                input        a, b, c, d,
                input [1:0]  select );
  always @*
    case (select)
      2'b00: y = a;
      2'b01: y = b;
      2'b10: y = c;
      2'b11: y = d;
    endcase
endmodule
```

**All possible 2-state values of select are specified as case item expressions**

**DC can determine this case statement is fully specified, even though there is no default assignment**

**Is this case statement fully specified in simulation? NO!**

# Example of a Partially Specified Decision

SUTHERLAND HDL
*Training Engineers to be HDL wizards*

SNUG SAN JOSE 2005

```
module mux3to1 (output reg   y,
                input        a, b, c,
                input [1:0]  select );
  always @*
    case (select)
      2'b00: y = a;
      2'b01: y = b;
      2'b10: y = c;
    endcase
endmodule
```

**DC cannot determine that this case statement is fully specified**
- **A select value of 2'b11 will not cause a branch to be executed**
- **In simulation, the y output will retain its previous value**

- If DC cannot determine that a decision statement is fully specified, latches are added to the hardware implementation
  - Ensures gate-level behavior will match RTL simulation behavior

**Note that a decision statement that is not fully-specified is one reason that synthesis will add latched logic to a design, but it is not the only reason**

# Parallel Decision Statements

SUTHERLAND HDL
*Training Engineers to be HDL wizards*

SNUG SAN JOSE 2005

- A parallel decision statement is one in which, for each decision control value, at most only one decision branch can be true
  - RTL simulation does not do parallel evaluations
    - `if...else` decisions and `case` statements are evaluated sequentially
    - Only the first matching branch is executed
  - If DC can determine that each case item expression is unique…
    - Then the decision expressions can be evaluated in any order
    - DC will optimize out the priority encoding of the RTL model
      - Fewer gates in the implementation
      - Shorter timing paths in the logic

**NOTE: DC only examines `case`/`casez`/`casex` statements to see if they are parallel decisions — DC does not examine `if...else` decision sequences**

# Example of a Parallel Decision Statement

SAN JOSE 2005

SUTHERLAND HDL
Training Engineers to be HDL wizards

- A parallel decision statement is one in which, for each decision control value, at most only one decision branch can be true

```
module mux4to1 (output reg  y,
                input        a, b, c, d,
                input [1:0] select );
  always @* begin
    case (select)
      2'b00: y = a;
      2'b01: y = b;
      2'b10: y = c;
      2'b11: y = d;
    endcase
  end
endmodule
```

**All case item expressions are unique (mutually exclusive)**

**DC can determine this case statement can be evaluated in parallel**

# Example of a Decision Statement that is not Parallel

SUTHERLAND HDL
Training Engineers to be HDL wizards

SNUG SAN JOSE 2005

- A parallel decision statement is one in which, for each decision control value, at most only one decision branch can be true

```
module interrupt_decode (output reg  something,
                         input [3:0] IRQ );
  always @* begin
    casez (IRQ) // use don't care bits when evaluating IRQ
      4'b???1: begin ... end  // process interrupt 0
      4'b??1?: begin ... end  // process interrupt 1
      4'b?1??: begin ... end  // process interrupt 2
      4'b1???: begin ... end  // process interrupt 3
    endcase
  end
endmodule
```

Note the use of casez and don't-care bits

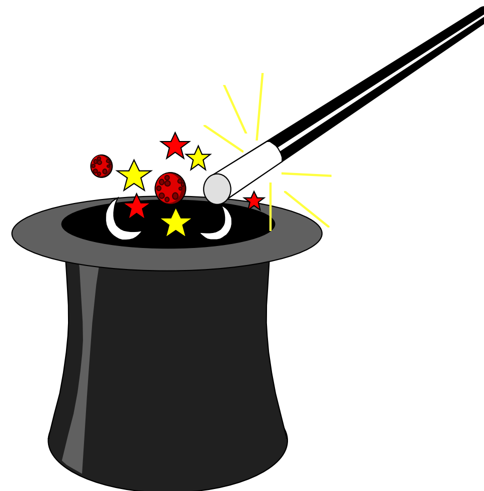
In simulation, if two or more IRQ bits are set at the same time, the lowest number bit will be the interrupt that is processed

DC will recognize that there is priority in how the RTL simulation will execute this casez statement, and add priority encoded logic to the gate-level implementation

# DC Does the Right Thing



- Synopsys DC synthesis compiler does the right thing (usually)
  - DC tries to generate a hardware implementation that approximates software simulation behavior
    - Correctly adds latches and/or priority encoded logic to match RTL simulation behavior
    - Correctly optimizes away latches and/or priority encoded logic when simulation does not depend on priority encoding or value storage



# Engineers are Smarter than Synthesis (or so we think)

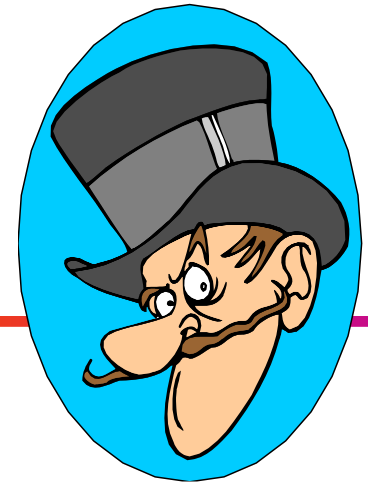


- Sometimes an engineer needs to override the default behavior of DC

  - In the 3-to-1 MUX example
    - The engineer might know a select value of 2'b11 will never happen
      - Therefore, the extra logic to latch the MUX output is not needed
  - In the interrupt decoder example
    - The engineer might know that two IRQ bits will never be set at the same time
      - Therefore, it is not necessary for the hardware to evaluate the IRQ bits in the same order as the software simulation model

SNUG SAN JOSE 2005

# The Villains of Synthesis: "full_case" & "parallel_case"



- DC provides two “pragmas” to control synthesis
  - The **full_case** pragma forces synthesis to leave out latches in a case statement that is not fully specified
  - The **parallel_case** pragma forces synthesis to leave out priority encoding in a case statement that is not parallel
- The pragmas are specified as a comment

```
case (select) // synopsys full_case
   ...
endcase
```

```
case (select) /* synopsys parallel_case */
  ...
endcase
```

**Either style of Verilog comment can be used**

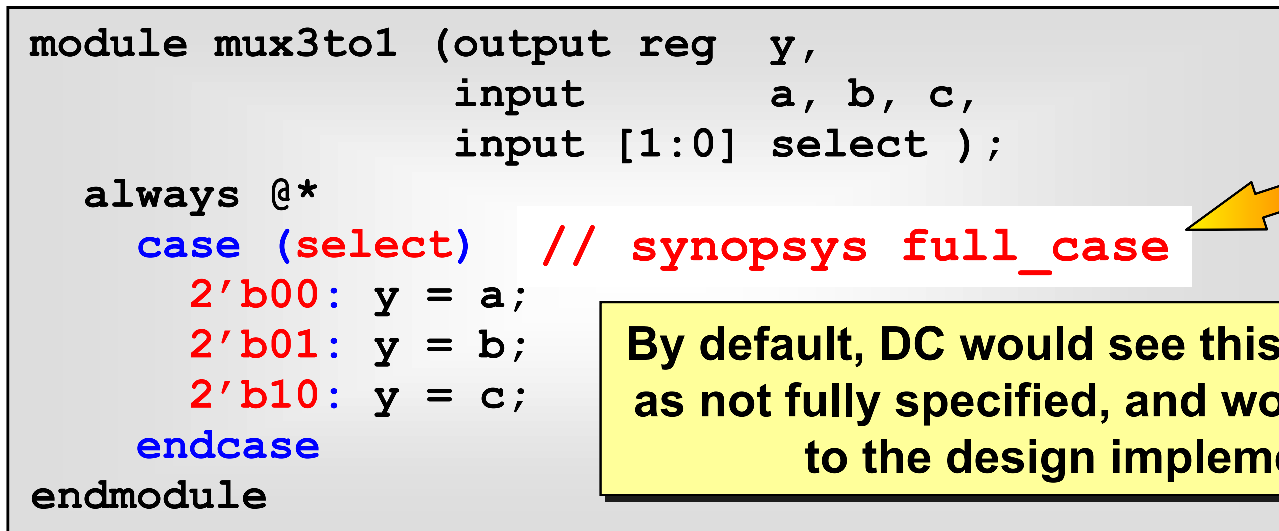
- **DC only supports the use of these pragmas with `case`, `casez` and `casex` statements**
- **The IEEE 1364.1 Verilog RTL synthesis standard provides another way to specify synthesis directives, using the Verilog attribute construct**

SNUG SAN JOSE 2005

# How the full_case Pragma Works

SUTHERLAND HDL
Training Engineers to be HDL wizards

- **full_case** instructs DC to ***assume*** that a `case`/`casez`/`casex` statement is fully specified
  - Case expression values that do not match a case item are ignored!



```
module mux3to1 (output reg  y,
                input       a, b, c,
                input [1:0] select );
  always @*
    case (select)  // synopsys full_case
      2'b00: y = a;
      2'b01: y = b;
      2'b10: y = c;
    endcase
endmodule
```

**By default, DC would see this case statement as not fully specified, and would add latches to the design implementation**

**The full_case pragma tells DC that a select value of 2'b11 will never happen, so treat the case statement as fully specified (do not add latches)**

**What will happen in the gate-level logic if the designer's assumption is wrong, and a select value of 2'b11 does occur?**

# Why full_case Is a Dastardly Villain

SUTHERLAND HDL
Training Engineers to be HDL wizards

SNUG SAN JOSE 2005

```
module mux3to1 (output reg  y,
                input       a, b, c,
                input [1:0] select );
  always @*
    case (select) // synopsys full_case
      2'b00: y = a;
      2'b01: y = b;
      2'b10: y = c;
    endcase
endmodule
```

**full_case tells synthesis to treat the case statement as if it were fully specified (even though it isn't)**

- What will happen if a select value of 2'b11 does occur?
  - RTL simulation will behave nicely (the y output does not change)
    - The designer's assumption that 2'b11 won't happen could go undetected!
  - The gate-level implementation will do something, but what?
    - ***A serious design flaw could exist in the actual hardware!***

- **DC will generate a warning for this example that the full_case pragma is used on a case statement that is not fully specified**
  - **Engineers will ignore the warning, because it only confirms what they assumed**

# How the parallel_case Pragma Works

- parallel_case tells DC to assume that all case item expressions are mutually exclusive, so priority encoding is not needed

```
module interrupt_decode (output reg  something,
                         input [3:0] IRQ );
  always @*
    casez (IRQ) // synopsys parallel_case
      4'b???1: begin ... end  // process interrupt 0
      4'b??1?: begin ... end  // process interrupt 1
      4'b?1??: begin ... end  // process interrupt 2
      4'b1???: begin ... end  // process interrupt 3
    endcase
endmodule
```

**Note the use of casez and don't-care bits**

**By default, DC would see that this case statement requires priority encoded logic**

**The parallel_case pragma tells DC that there will NEVER be more than one case item expressions true at the same time, so priority encoding is not needed**

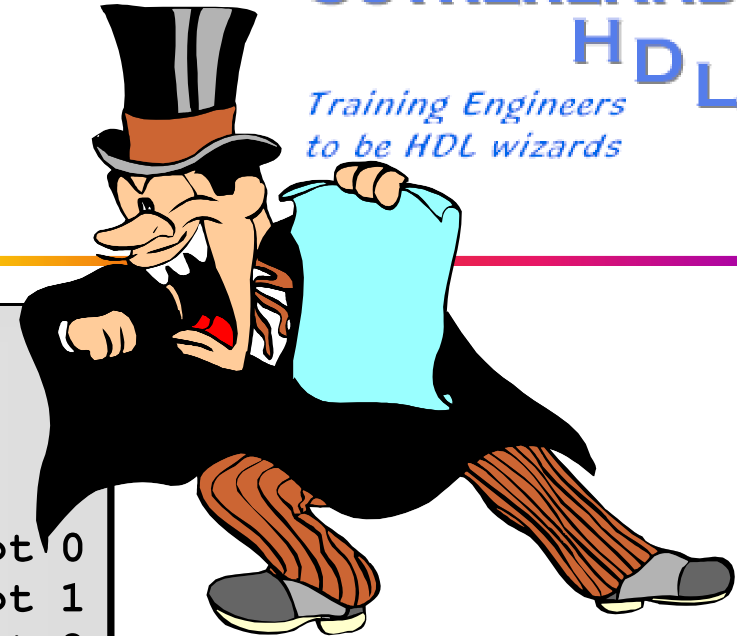
**What will happen in the gate-level logic if the designer's assumption is wrong, and multiple IRQ bits are set at the same time?**

SNUG SAN JOSE 2005

SUTHERLAND HDL
Training Engineers to be HDL wizards

SNUG SAN JOSE 2005

# Why parallel_case Is an Evil Villain




```
module interrupt_decode (output reg  something,
                         input [3:0] IRQ );
  always @*
    casez (IRQ) // synopsys parallel_case
      4'b???1: begin ... end  // process interrupt 0
      4'b??1?: begin ... end  // process interrupt 1
      4'b?1??: begin ... end  // process interrupt 2
      4'b1???: begin ... end  // process interrupt 3
    endcase
endmodule
```

**parallel_case tells synthesis to assume that all case item expressions are mutually exclusive**

- What will happen if two or more IRQ bits are set at the same time?
  - RTL simulation will behave nicely (only the first matching branch is executed)
    - The designer's assumption about IRQ could go undetected!
  - The gate-level implementation will do something, but what?
    - ***A serious design flaw could exist in the actual hardware!***

- **DC will generate a warning for this example that the parallel_case pragma is used on a case statement where more than one case item expression could be true**
  - **Engineers will ignore this warning, because it only confirms what they assumed**

# Who Is Going To Save the Design Engineer?

SAN JOSE 2005

SUTHERLAND HDL
*Training Engineers to be HDL wizards*

# unique and priority Decisions, The Designer's Heroes!

- SystemVerilog adds decision modifiers
  - New keywords, **unique** and **priority**
  - Specified before an **if**, **case**, **casez** or **casex** keyword

```
always @*
  unique case (select)
    2'b00: y = a;
    2'b01: y = b;
    2'b10: y = c;
  endcase
```

```
always @*
 priority if (IRQ[0]) // process interrupt 0
      else if (IRQ[1]) // process interrupt 1
      else if (IRQ[2]) // process interrupt 2
      else if (IRQ[3]) // process interrupt 3
      else             // process no interrupt
```

- The **unique** and **priority** decision modifiers provide
  - All of the benefits of full_case/parallel_case synthesis pragmas
  - None of the dangers associated with synthesis pragmas
  - Additional benefits that full_case and parallel_case do not have

# What Makes the unique Decision Modifier a Hero

SUTHERLAND HDL

*Training Engineers to be HDL wizards*

SNUG SAN JOSE 2005

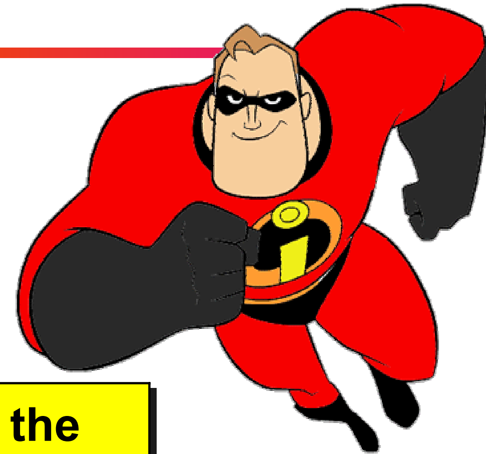
- `unique` instructs tools to treat the selection items in a series of decisions as unique values (mutually exclusive)
  - `unique` is a keyword in the Verilog source code
    - full_case/parallel_case pragmas are hidden inside a comment
  - `unique` affects all tools (simulation, synthesis, formal, etc.)
    - full_case/parallel_case pragmas only affect synthesis
  - `unique` can be used with both `case` and `if...else` decisions
    - full_case/parallel_case can only be used with `case` statements
  - `unique` will generate simulation warning messages if misused
    - full_case/parallel_case pragmas are ignored by simulation
    - ✓ Simulation warning if more than one branch in a `case` or `if...else` decision sequence could be true at the same time
    - ✓ Simulation warning if no branch of a decision sequence is executed

# unique Saves a Design that uses a 3-to-1 MUX!

SNUG San Jose 2005

Sutherland HDL — Training Engineers to be HDL wizards

```
module mux3to1 (output reg   y,
                input        a, b, c,
                input [1:0]  select );
  always @*
   unique case (select)
      2'b00: y = a;
      2'b01: y = b;
      2'b10: y = c;
    endcase
endmodule
```



**What will happen in the gate-level logic if a select value of 2'b11 does occur?**

**With the villainous full_case pragma:**

- Synthesis will ignore the undefined value of select (2'b11)
  - Should the value occur, the gate-level implementation will have a design flaw
- Simulation will ignore the pragma
  - Should the select value 2'b11 occur, simulation will run without warnings
  - A design error could go undetected!

**With the heroic unique decision modifier:**

- Synthesis will ignore the undefined value of select (2'b11)
  - Should the value occur, the gate-level implementation will have a design flaw
- Simulation will *not* ignore the modifier
  - Should the select value 2'b11 occur, simulation will have run-time warnings
  - The design error will be detected!

**The unique decision modifier has saved the design (and the designer!)**

# unique Saves a Design with an Interrupt Decoder!

SUTHERLAND HDL
Training Engineers to be HDL wizards

SNUG SAN JOSE 2005

```
module interrupt_decode (output reg  something,
                         input [3:0] IRQ );
  always @*
   unique casez (IRQ)
      4'b???1: begin ... end  // process interrupt 0
      4'b??1?: begin ... end  // process interrupt 1
      4'b?1??: begin ... end  // process interrupt 2
      4'b1???: begin ... end  // process interrupt 3
    endcase
endmodule
```

**What will happen in the gate-level logic if two or more IRQ bits are set at the same time?**

With the dastardly parallel_case pragma:

- Synthesis will assume that two or more IRQ bits will never be set at the same time
  - Should two bits be set, the gate-level implementation will have a design flaw
- Simulation will ignore the pragma
  - Should multiple IRQ bits be set, simulation will run without warnings
  - The design error could go undetected!

With the heroic unique decision modifier:

- Synthesis will assume two or more IRQ bits will never be set at the same time
  - Should two bits be set, the gate-level implementation will have a design flaw
- Simulation will *not* ignore the modifier
  - Should multiple IRQ bits be set, simulation will have run-time warnings
  - The design error will be detected!

**The unique decision modifier has saved the design (and the designer!)**

# What Makes the priority Decision Modifier a Hero

- `priority` instructs tools that each selection item in a series of decisions must be evaluated in the order they are listed
  - The order of evaluation must be maintained by all tools
  - `priority` is a keyword in the Verilog source code
    - Synthesis pragmas are hidden inside a comment
  - `priority` affects all tools (simulation, synthesis, formal, etc.)
    - Synthesis pragmas only affect synthesis
  - `priority` can be used with both `case` and `if...else` decisions
    - Synthesis can only be used with `case` statements
  - `priority` will generate simulation warning messages if misused
    - Synthesis pragmas are ignored by simulation
    - ✓ Simulation warning if a `case` or `if...else` decision sequence is entered, and no branch of a decision sequence is executed

SNUG SAN JOSE 2005

SUTHERLAND HDL
Training Engineers to be HDL wizards

# priority Saves a Design with an Interrupt Decoder!

```
module interrupt_decode (output reg  something,
                         input [3:0] IRQ );
  always @*
   priority case (IRQ)
      4'b0000: begin ... end  // no interrupts
      4'b0001: begin ... end  // process interrupt 0
      4'b0010: begin ... end  // process interrupt 1
      4'b0100: begin ... end  // process interrupt 2
      4'b1000: begin ... end  // process interrupt 3
    endcase
endmodule
```

**What will happen in the gate-level logic if two or more IRQ bits are set at the same time?**

**Without the priority decision modifier:**

- Synthesis will detect that the case item expression values are mutually exclusive
  - The priority might be removed and the interrupts evaluated in parallel
- Simulation will simulate with priority
  - If multiple IRQ bits are set, no warning will occur
  - A design error could go undetected!

**With the heroic priority decision modifier:**

- Synthesis will ***not*** optimize out the RTL model priority of the select values
  - The interrupts will be evaluated with the same priority as the RTL model
- Simulation will simulate with priority
  - If multiple IRQ bits are set, a run-time warning will occur
  - The design error will be detected!

# Comparing unique to Synthesis Pragmas

- For synthesis
  - `unique case()` (or casez/casex) is the same as specifying both `full_case` and `parallel_case`
    - Synthesis will optimize a `unique case` as if it were both fully specified and all case select items are mutually exclusive
  - `unique if()` does not exist in synthesis pragmas
    - full_case and parallel_case can only be used with case/casez/casex
- For simulation
  - `unique` can be used to verify that the design will function correctly with the synthesis optimizations
    - `full_case` and `parallel_case` do not affect simulation
  - `unique` can be used to verify the designer's assumption for both `case` and `if...else` decision sequences

# Comparing priority to Synthesis Pragmas

SNUG SAN JOSE 2005

SUTHERLAND HDL
Training Engineers to be HDL wizards

- For synthesis
  - full_case has *some* of the same functionality as **`priority`**
    - Both indicate that the **`case`** statement can be treated as a fully specified case statement
  - **`priority`** does more than the full_case pragma!
    - The **`priority`** modifier enforces the intended order of multiple branch decision statements
    - There is no synthesis pragma (or combination or pragmas) that requires synthesis to maintain the priority of a decision statement
  - **`priority`** can also be used with **`if...else`** decision sequences
- For simulation
  - **`priority`** can be used to verify the designer's assumption for both **`case`** and **`if...else`** decision sequences
    - Gives automatic warnings if no branch of a decision is executed

# Coding Guidelines

1) Use the **`unique`** decision modifier
   - If each branch of the decision is controlled by a different value
   - *And*, two or more branches should never be true at the same time
2) Use the **`priority`** decision modifier
   - If it is possible for more than one branch control to be true at the same time
   - *But*, there should never be a situation where no branch is executed
3) Do *not* specify a decision modifier when a decision sequence does not need to be fully specified
   - When modeling a latch
   - When modeling a sequential device like a flip-flop
   - When a default assignment to all combinatorial outputs has been made before the decision sequence
4) Never, ever, at any time, use the dastardly synthesis full_case and parallel_case synthesis pragmas

# Conclusions

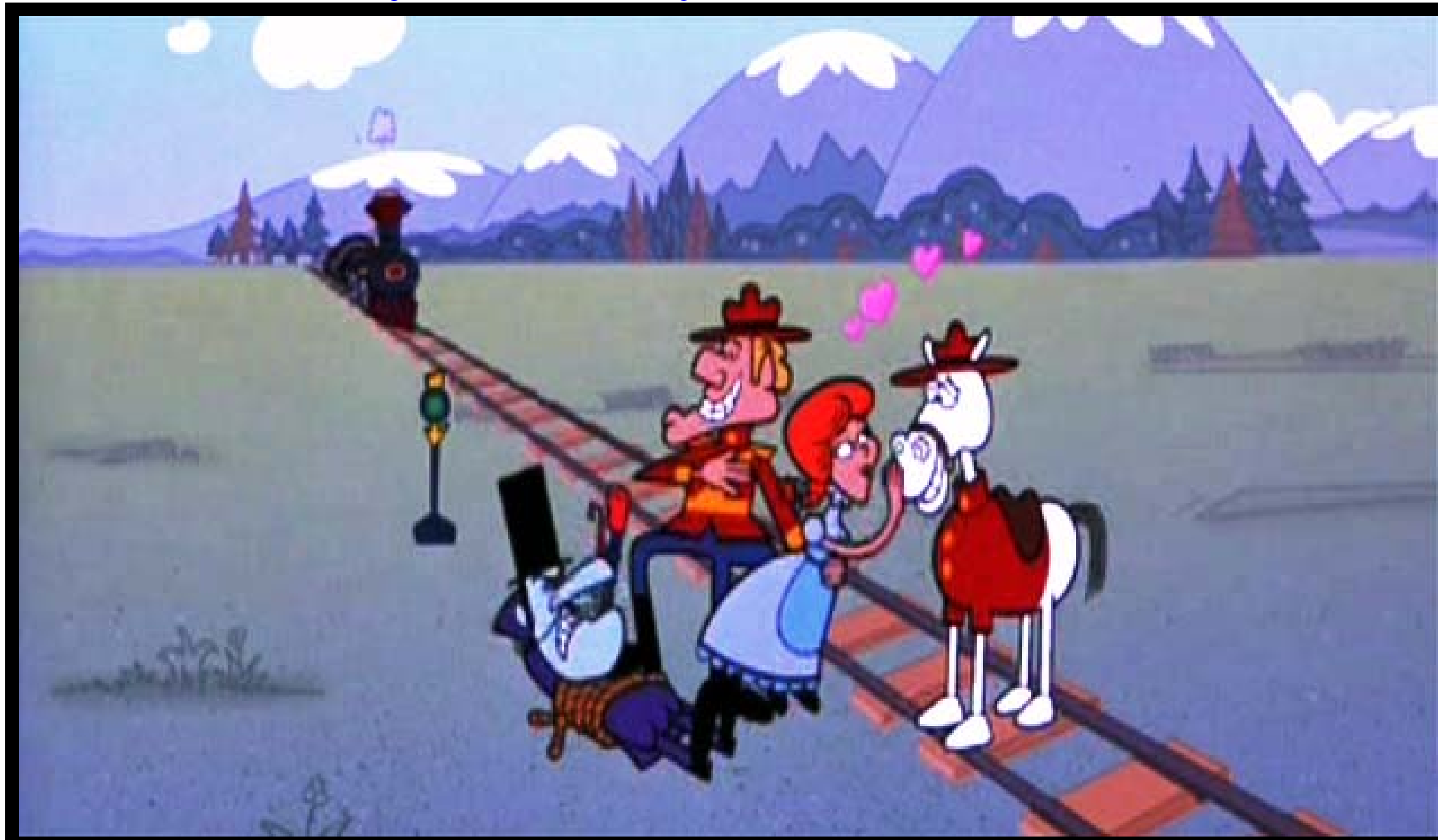
- The `unique` and `priority` decision modifiers are heroes that can prevent difficult to find design errors!
  - ✓ Help to ensure that designs will function correctly, both before and after synthesis
  - ✓ Affect all Verilog design tools, and not just synthesis
  - ✓ Document the design engineer's expectations for how decision statements will execute
  - ✓ Can be used with both `if...else` decisions and `case`, `casez` or `casex` statements
  - ✓ `priority` enforces the intended order of multiple branch decision statements (there is no synthesis pragma equivalent)
  - ✓ Run-time simulation warnings help ensure that the designer’s assumptions about each decision statement are indeed correct

# Questions?



## SystemVerilog Saves the Day, the Evil Twins are Defeated

## "unique" and "priority" are the new heroes!



# An Example of When full_case and parallel_case Are Needed

SUTHERLAND HDL
Training Engineers to be HDL wizards

SNUG SAN JOSE 2005

```
module FSM (...);
  reg [3:0] state;  // 4-bit wide state vector

  always @*
    case (1'b1)  // synopsys full_case parallel_case
      state[0]: begin ... end  // set state 1 output values
      state[1]: begin ... end  // set state 2 output values
      state[2]: begin ... end  // set state 3 output values
      state[3]: begin ... end  // set state 1 output values
    endcase
endmodule
```

case expression

case items

One-hot state machine decoder, using a "reverse case statement"
- Only one bit at a time will be set in the state vector (1-hot)
- The case expression and case item expressions are reversed
  - The value to be detected is the case expression
  - Bits of the state vector are the case item expressions

Without the full_case and parallel_case pragmas, synthesis will not optimize this model
- The case statement is not full — a state value with no bits set is not detected
- The case statement is not parallel — it is possible for more than one bit to be set

# Case Study:
# The Villain Wins!

SUTHERLAND HDL
*Training Engineers to be HDL wizards*

SNUG SAN JOSE 2005

- In his paper *"The Evil Twins of Synthesis"* (SNUG Boston, 1999), Cliff Cummings cites a case where:
  - The design engineer added full_case and parallel_case pragmas to all `case` statements in the design
    - Wanted a smaller, faster gate-level implementation of the design
  - The design had a `case` statement that depended on the priority in which the case items were evaluated
    - In RTL simulation, the design appeared to work correctly
      - Simulation always simulates case statements with priority
  - Synthesis did what the (clever?) design engineer told it to do, and left out the priority encoded logic
    - The ASIC did not work correctly, and had to be re-spun
    - Cost the company many thousands of dollars in actual costs
    - Cost the company untold dollars in project delays