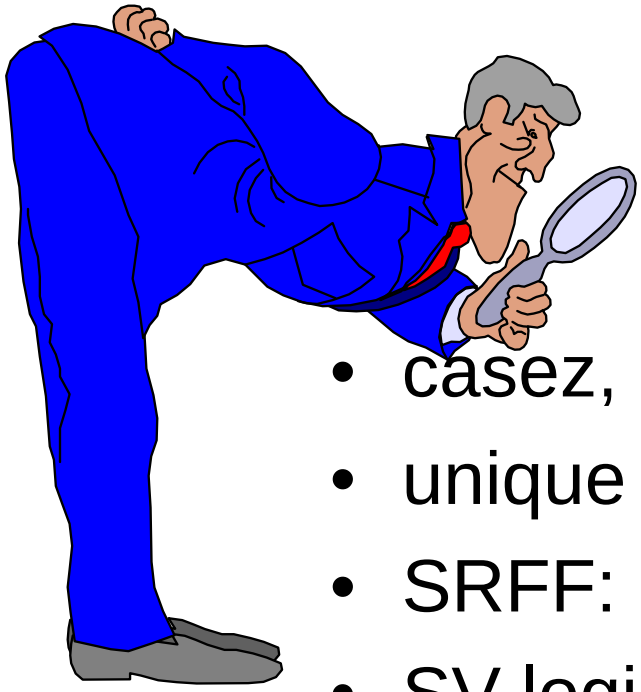
Synopsys Users Group
SILICON VALLEY 2012

Yet Another Latch and Gotchas Paper

Don Mills
Microchip Technology, INC
Chandler, AZ

Discussion Topics



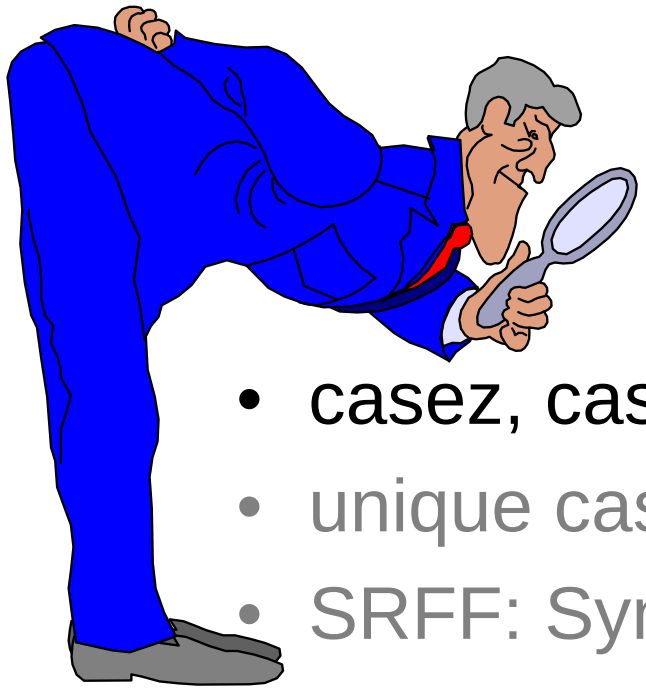
- casez, casex, case inside
- unique case & priority case still allow latches
- SRFF: Synopsys RTL coding bug work-arounc
- SV logic type NOT, what is SV logic really?

A little history

Synopsys Users Group
SILICON VALLEY 2012

- Past papers provide details on these topics
  - Clifford Cummings, *"'full_case parallel_case', the Evil Twins of Verilog Synthesis"*
  - Clifford Cummings, "*SystemVerilog's priority & unique—A Solution to Verilog's 'full_case' & parallel_case' Evil Twins!*"
  - Stuart Sutherland, "*SystemVerilog Saves the Day—the Evil Twins are Defeated! 'unique' and 'priority' are the new Heroes*"
  - Don Mills and Clifford Cummings, *"RTL Coding Styles That Yield Simulation and Synthesis Mismatches"*
  - Stuart Sutherland and Don Mills, *"Standard Gotchas, Subtleties in the Verilog and SystemVerilog Standards That Every Engineer Should Know"*
  - *Shalom Bresticker, "Just When You Thought it Was Safe to Start Coding Again…Return of the SystemVerilog Gotchas"*
  - Don Mills, *"Being Assertive With Your X"*

Discussion Topics



- casez, casex, case inside
- unique case & priority case still allow latches
- SRFF: Synopsys RTL coding bug work-around
- SV logic type NOT, what is SV logic really?

Case Statement Definitions

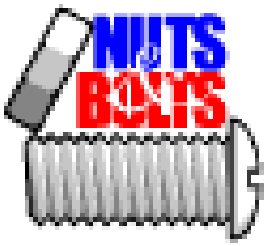
Synopsys Users Group
SILICON VALLEY 2012

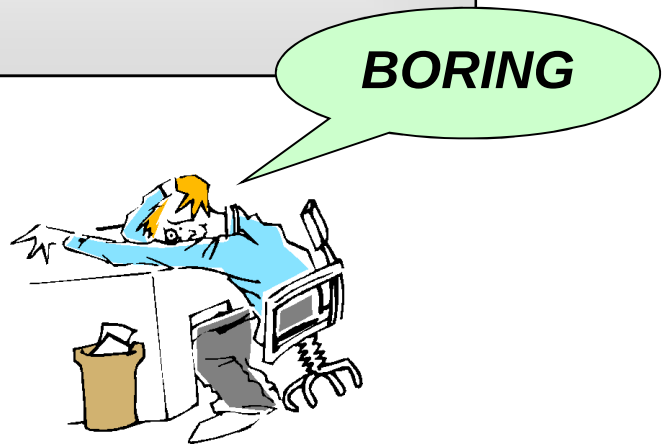
Case expression

Case items

Default case item

```
always_comb begin
    case (sel)
      2'b00 : out = a;
      2'b01 : out = b;
      2'b10 : out = c;
      2'b11 : out = d;
    default : out = 'X;
    endcase
  end
```





Case statement

```
always_comb begin
    case (sel)
      2'b00 : out = a;
      2'b01 : out = b;
      2'b10 : out = c;
    endcase
  end
```

3:1 mux – case will generate latches – *not covering 2'b11 state in simulation or synthesis*

```
always_comb begin
    case (sel)
      2'b00 : out = a;
      2'b01 : out = b;
      2'b10 : out = c;
      2'b11 : out = d;
    endcase
  end
```

4:1 mux – case will not generate latches in synthesis and usually not in simulation

what is the simulation output when sel has X's or Z's in it?

Case statement with default

Synopsys Users Group
SILICON VALLEY 2012

- Propagating the X is common
  - In simulation, the X must be visible
  - Must trace back to the source of the X

```
always_comb begin
    case (sel)
      2'b00 : out = a;
      2'b01 : out = b;
      2'b10 : out = c;
      2'b11 : out = d;
    default : out = 'X;
    endcase
  end
```

4:1 mux – case will not generate latches in synthesis and usually not model latches in simulation

what is the simulation output when sel has X's or Z's in it?

if / else vs conditional operator for X prop

- if / else is known to block X prop if sel is unknown
- Many use the conditional operator (thinking it will always propagate X's) – wrong!

```
always_comb begin
  if (sel)
    out = a;
  else
    out = b;
end
```

sel	a	b	out
0	dead	beef	dead
1	dead	beef	beef
X	dead	beef	beef

```
assign out = sel ? a : b;

     OR

always_comb
  out = sel ? a : b;
```

sel	a	b	out
0	dead	beef	dead
1	dead	beef	beef
X	dead	beef	xexx
X	dead	dead	dead

BOTH methods will prevent X propagation

casex / casez statement

Synopsys Users Group
SILICON VALLEY 2012

```
always_comb begin
    {memce0, memce1, cs} = '0;
    casex ({addr, en})
      3'b101 : memce0 = '1;
      3'b111 : memce0 = '1;
      3'b0?1 : cs = '1;
    endcase
  end
```

casex – X or Z or ? (another symbol for Z) are don't cares in case items or case expression

```
always_comb begin
    {memce0, memce1, cs} = '0;
    casez ({addr, en})
      3'b101 : memce0 = '1;
      3'b111 : memce0 = '1;
      3'b0?1 : cs = '1;
    endcase
  end
```

casez – Z or ? (another symbol for Z) are don't cares in case items or case expression

What if en or addr has an X value

Previous guideline – use casez over casex – better guideline coming

case inside statement with default

- case inside allows case item don't cares using X, Z or ?
- X, Z in case expression is literal rather than don't care

```
always_comb begin
    {memce0, memce1, cs} = '0;
    case ({addr, en}) inside
      3'b101 : memce0 = '1;
      3'b111 : memce0 = '1;
      3'b0?1 : cs = '1;
     default : begin
               // synthesis translate_off
                 $display("Error in case expression");
               // synthesis translate_on
               end
    endcase
  end
```

A case inside will simulate the exact way the casex synthesizes

casex immediate assertions

Synopsys Users Group
SILICON VALLEY 2012

Assertions allow
- **casex** and **casez** to be same
- Localized X finding
- Can be disabled

```
always_comb begin
  assert (!$isunknown({addr, en}))
    else $error("%m : case_sel = X");
  {memce0, memce1, cs} = '0;
  casex ({addr, en})
    3'b101 : memce0 = '1;
    3'b111 : memce0 = '1;
    3'b0?1 : cs = '1;
  endcase
end
```

```
always_comb begin
  assert (!$isunknown({sel}))
    else $error("%m : case_sel = X");
  case (sel)
    2'b00 : out = a;
    2'b01 : out = b;
    2'b10 : out = c;
    2'b11 : out = d;
  endcase
end
```

NOTE: use this same assertion method for if statements

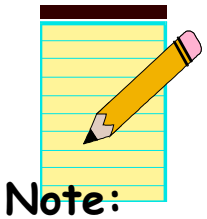
Discussion Topics

- casez, casex, case inside
- unique case & priority case still allow latches
- SRFF – Synopsys coding error revealed!
- SV logic type NOT, what is SV logic really?

Synthesis Directives

Synopsys Users Group
SILICON VALLEY 2012

- Lots of papers on directives and SV enhancements




```
full_case
parallel_case
```

Often represented the code differently in synthesis than in simulation

Should only be used with “inverse” case statement type coding

SV 2005 case enhancements

Synopsys Users Group
SILICON VALLEY 2012

- Lots of papers on SV Enhancements

```
unique case
priority case
```

- **Designed to provide same functionality in simulation and synthesis**
- **Provides simulation warnings of possible latch type code**
- **Unfortunately – does not issue warning for all latch conditions**

unique case with latches

- unique case only looks at case expression / case items
  - No overlapping conditions
  - A condition matches each time the case is entered

```
always_comb begin
  unique case (sel)
     2'b00 : begin out1 = a1;
                   out2 = a2;
             end
     2'b01 : out2 = b;
     2'b10 : out1 = c;
   default : begin out1 = a1;
                   out2 = a2;
             end
  endcase
end
```

This case statement has unique case and a default but will still generate latches with *no warnings* until synthesis reports

unique case with external default

- unique case synthesizes same as full_case

- Defaults outside the case may be ignored by synthesis

```
always_comb begin
  out1 = a1;
  out2 = a2;
  unique case (sel)
    2'b00 : begin out1 = a1;
                  out2 = a2;
            end
    2'b01 : out2 = b;
    2'b10 : out1 = c;
  endcase
end
```

full_case (*and unique case*) assumes *all the content for the case is defined in the case* and all other conditions are don't cares for synthesis

Two possible solutions to always prevent latches

Synopsys Users Group
SILICON VALLEY 2012

```
always_comb begin
  unique case (sel)
      2'b00 : begin out1 = a1;
                    out2 = a2;
              end
      2'b01 : begin out1 = a1;
                    out2 = b;
              end
      2'b10 : begin out1 = c;
                    out2 = a2;
              end
    default : begin out1 = a1;
                    out2 = a2;
              end
  endcase
end
```

Solution1: Define all outputs for all case conditions – messy with lots of outputs

```
always_comb begin
  out1 = a1;
  out2 = a2;
  case (sel)
      2'b01 : out2 = b;
      2'b10 : out1 = c;
  endcase
end
```

Solution 2: Define default outputs prior to any conditional logic.

Small code – no latches

unique0 case solution (if it was only supported)

Synopsys Users Group
SILICON VALLEY 2012

- **`unique0 case`** only looks for case select / case items
  - No overlapping conditions
  - Does not require a match when the case is entered

```
always_comb begin
  out1 = a1;
  out2 = a2;
  unique0 case (sel)
     2'b01 : out2 = b;
     2'b10 : out1 = c;
  endcase
end
```

This case statement has `unique0 case` and `defaults` before the case

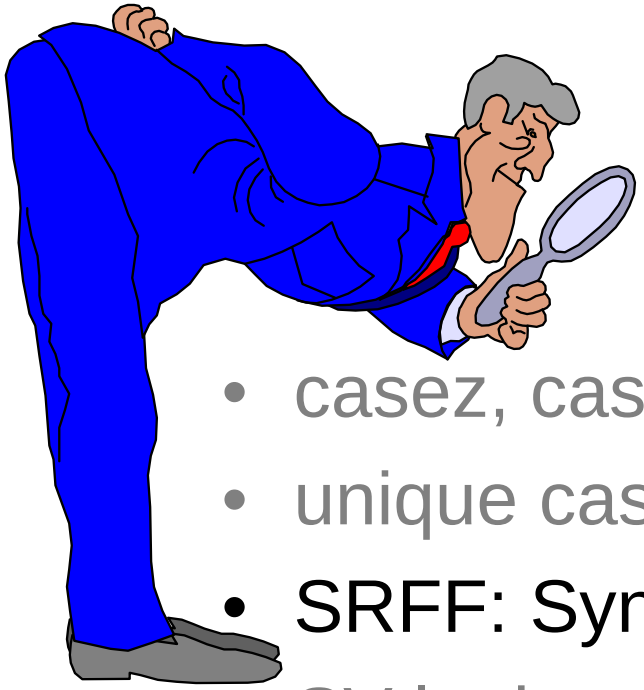
No latches
No priority encoder

Currently, `unique0` is not supported by simulation or synthesis BUMMER!!!!!

(unique0 IS REALLY JUST PARALLEL CASE)

MICROCHIP

Discussion Topics

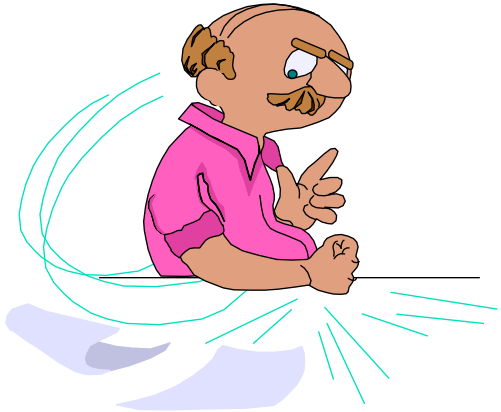


- casez, casex, case inside
- unique case & priority case still allow latches
- SRFF: Synopsys RTL coding bug work-around
- SV logic type NOT, what is SV logic really?

SRFF – RTL vs Synthesis Model

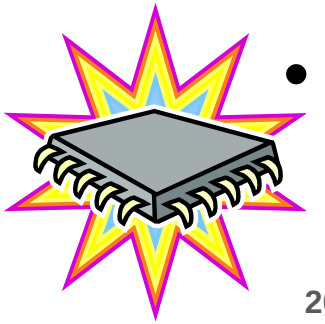
Synopsys Users Group
SILICON VALLEY 2012

- Are asynchronous Set/Reset Flip-Flops used?



YES

- Asynchronous Set/Reset Flip-Flops manage IC fuses for chip configuration
  - Google search **microprocessor fuses**



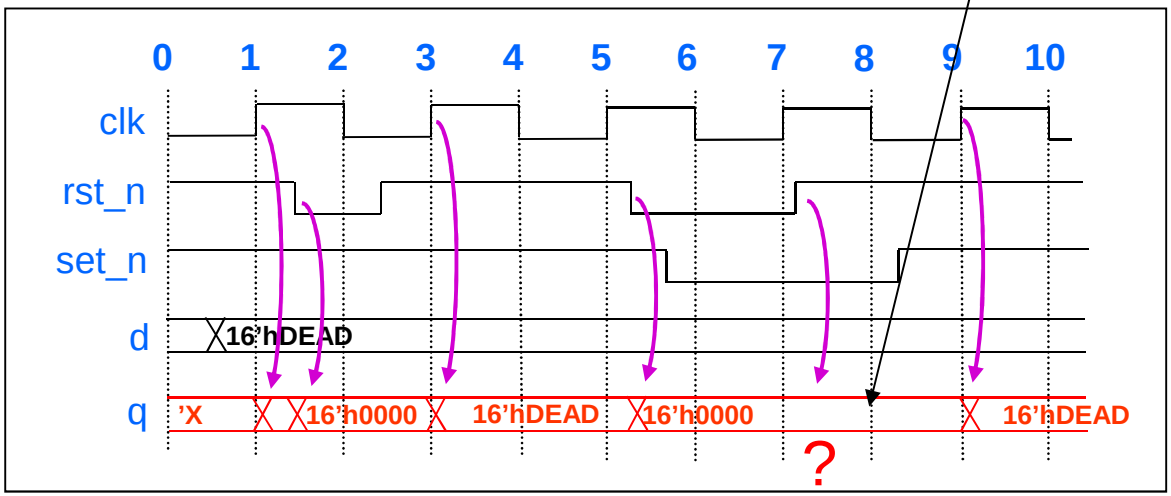
SRFF: Synthesis required coding style

```
always_ff @(   posedge clk
            or negedge rst_n
            or negedge set_n
           );
  if       (!rst_n) q <= '0;
  else if (!set_n) q <= '1;
  else              q <=  d;
```

NOTE: `rst_n` has priority over `set_n`

The `set_n` has no effect on simulation results.

This code will synthesize correctly but fails in this simulation corner case



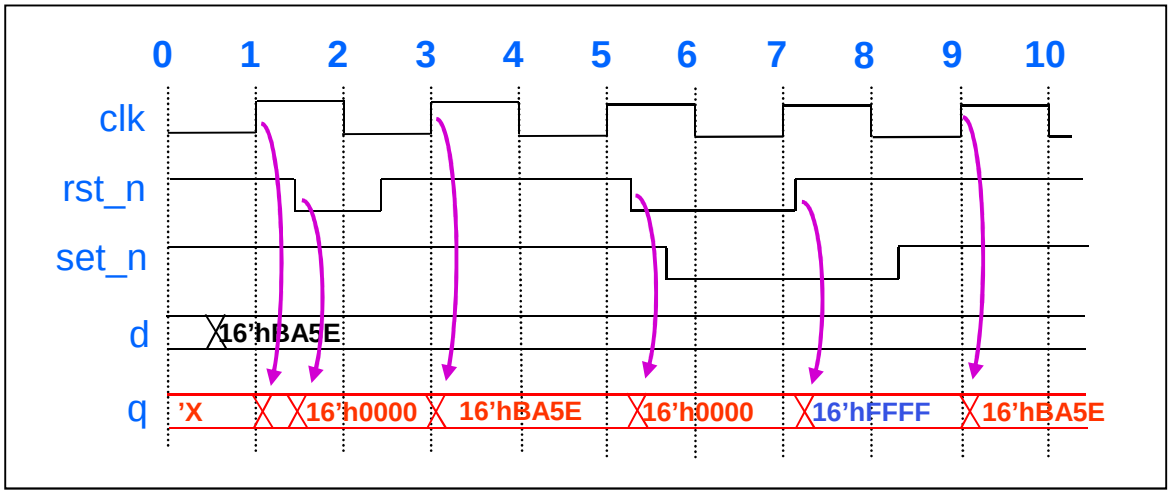
SRFF: Simulation model for Synthesis

Synopsys Users Group
SILICON VALLEY 2012

```
always_ff @(   posedge clk
            or negedge rst_n
            or negedge set_n
`ifndef SYNTHESIS
            or posedge (rst_n & ~set_n)
`endif
            );
  if       (!rst_n) q <= '0;
  else if (!set_n) q <= '1;
  else              q <=  d;
```

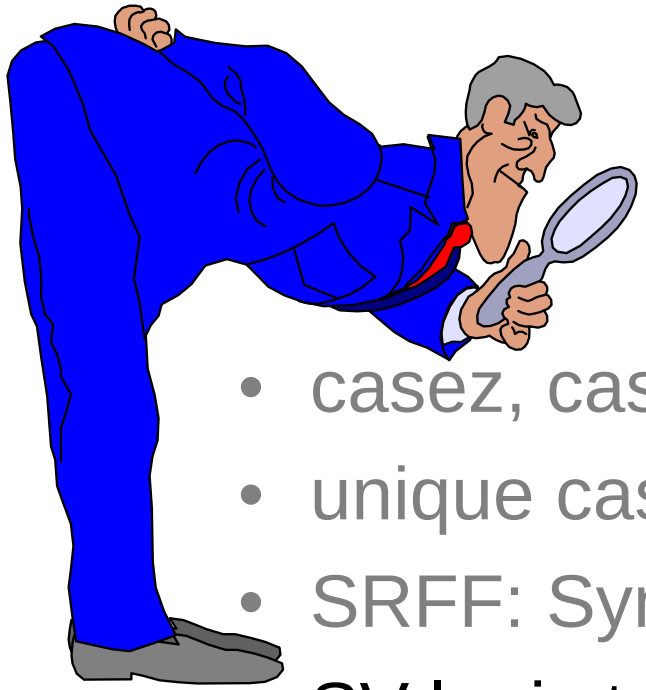
NOTE: `rst_n` has priority over `set_n`

This code will simulate and synthesize correctly



MICROCHIP

Discussion Topics



- casez, casex, case inside
- unique case & priority case still allow latches
- SRFF: Synopsys RTL coding bug work-around
- SV logic type NOT, what is SV logic really?

SV Logic – redefined in 1800-2009

- IEEE 1800-2005
  - `logic` and `bit` **variable** declaration
- IEEE 1800-2009
  - `logic` and `bit` define a **value set:**

    4 state or 2 state
- IEEE 1800-2009 `logic` and `bit` still imply **variable** types…
  - except for `input` and `inout` ports

DOES THIS MATTER?
Why do I care?

IEEE 1800-2009 `logic` gotcha - inputs

- Inputs default to **`wire logic`** type
- You must specify **`var`** if you want a variable type

```
module M (input              a,  //infers wire logic
          input        logic b,  //infers wire logic
          input var          c,  //infers var  logic

          output             d,
          output wire        e,
          output tri         f,
          output       logic g,

          output             h,
          output       logic k);
```

Currently, `presto` still reads by the 1800-2005 rules:
`logic` by itself implied var logic

Why do I care??

Variables may only have a single source.

Variable inputs may limit simulation optimization. Possibly an issue for large designs

IEEE 1800-2009 `logic` gotcha - inouts

- What's the difference between `input` and `inout`?
- What's the difference between `wire` and `tri`?

```
module M (input             a,   //infers wire logic
          input       logic b,   //infers wire logic
          input var   logic c,

          inout             d,   //infers wire logic
          inout wire        e,   //infers wire logic
          inout tri         f,   //infers wire logic
          inout       logic g,   //infers wire logic

          output            h,
          output      logic k);
```

Inside the simulator, there may be an implementation difference between input and inout ports which will affect speed for "large" designs.

Externally, there is not function difference.

IEEE 1800-2009 `logic` gotcha - outputs

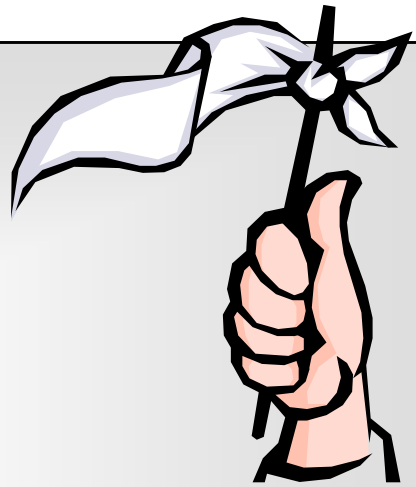
Synopsys Users Group
SILICON VALLEY 2012

- logic outputs default to variable **`logic`**

```
module M (input              a,
          input        logic b,
          input var    logic c,

          inout              d,
          inout wire         e,
          inout tri          f,
          inout        logic g,

          output             h,  //infers wire logic
          output       logic k); //infers var  logic
```



Personal preferences

Synopsys Users Group
SILICON VALLEY 2012

- Option 1 – (*Cliff Cummings - reviewer*)
  - Use **`logic`** everywhere
    - except when a signal is multi-driven, then use **`wire`**

- Option 2 – (*Don Mills*) *The VHDL coming out in me*
  - Use **`logic`** everywhere
    - except when a signal is multi-driven, then use **`tri`** (self documenting)
    - To be pure in intent, should now use **`var logic`** for inputs (but probably won't)

Conclusion

Synopsys Users Group
SILICON VALLEY 2012

- casez, casex, case inside
  - Use immediate assertion with casex or case inside
- unique case & priority case still allow latches
  - Use defaults prior to any conditional logic to prevent latches (Use unique0 when supported)
- SRFF – Synopsys coding error revealed!
  - Add simulation fix to sensitivity list
- SV logic type “NOT”, what is SV logic really?
  - Use logic everywhere / except when a signal is multi-driven – then use tri (or wire)