Being Assertive With Your X

(SystemVerilog Assertions for Dummies: version 2013)

Don Mills

Microchip Technology Inc,

Chandler, AZ, USA

don.mills@microchip.com

mills@lcdm-eng.com

Don Mills
MICROCHIP

Outline

- The Origin of X
- The Big Lie - If you don’t see it, it isn’t there.
- The Old-Fashioned solution
- The tool kludge being promoted today
- Using SystemVerilog assertions to seek out your X
- SystemVerilog assertion bind
- Conclusions and Recommendations
- **Mentor’s Best Kept Secret**

The Origin of X

Don Mills
MICROCHIP

- Uninitialized Variables (4-state types)
- Flip-flop data path chain
  - Follower Flops
  - Uninitialized flops and latches
- Multiple drivers on a wire
  - without a tri-state driver
- Direct X assignment
- Undriven wires
- Floating input ports
- Gate level X's
  - timing violations
  - UDP
- UPF – low power mode
- Out of range bit-selects and indices

The Big Lie

- Designers focus on functionality
- Seldom is RTL coded with integrity checkers
- Code that blocks or hides X's :
  - **`if/else`** statements
  - **`case/casez/casex/case inside`** statements
  - combinational logic
    - and/or logic
  - ternary operator " **`? :`** "

if/else statements X-hiding

- One of the most common X-hiding statement is an **if/else** statement

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

The **if** will be true if **sel** is true (1'b1)

The **else** condition will execute for all other values of **sel**
(1'b0, 1'bX, 1'bZ)

case statements X-hiding

- For standard **case** statements (***NOT casez, casex, or case inside***)

```
logic [1:0] sel;
logic a, b, c, d, out;

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

when the **case** expression has an 1'bX (or a 1'bZ) it's value will be lost and the **case** statement will hold it's previous value

casex, casez, case inside statements X-hiding

Don Mills
MICROCHIP

- **`casex, casez, case inside`** statements apply “don’t-care” mapping to X’s and Z’s.

```
logic [1:0] sel;
logic a, b, c, d, out;

always_comb
  casex (sel)
    2'b00 : out = a;
    2'b01 : out = b;
    2'b1X : out = c;
    2'bZ1 : out = d;
  endcase
```

When the **`case`** expression has an 1’bX (or a 1’bZ) it will be treated as a don’t care for **`casex`** and **`casez`**.

When the **`case`** items have X’s or Z’s they will be treated as don’t cares for **`casex`**, **`casez`**, **`case inside`**.

(lots of SV rules not discussed here)

operator for X-propagation

Don Mills

MICROCHIP

- 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

and/or expression X-hiding

- Anything and'ed with a low will output a low
- Anything or'ed with a high will output

```
logic a, b, c, d, out1, out2;

assign out1 = a & b;
assign out2 = c | d;
```

```
logic a, b, c, d, out1, out2;

and (out1, a, b);
or  (out2, a, b);
```

The Old-Fashioned Solution

- Add extra condition checking in the middle of the RTL functional description
- Add **`$display`** statements in RTL functional code
  - *Hide these display statement from synthesis using pragmas*
- Very cumbersome to code and maintain and therefore rarely used

Verilog Manual Assertion

Don Mills
MICROCHIP

```
always_comb
  if (sel == 1'b1)
    out = a;
  else
    //pragma translate off
    if (sel == 1'b0)
    //pragma translate on
    out = b;
    //pragma translate off
    else // sel == 1'bX or 1'bZ
      begin
        out = 'x;
        $display(%m "assert:
          sel X or Z at time %d",
          $time);
      end
    //pragma translate on
```

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

Changing the rules?

- Some simulators provide an X-propagation/X-optimization option
  - The X-propagation mode: changes the RTL rules and propagates an X out even if a and b are known
  - The X-optimization mode: If all the data inputs va

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

sel	a	b	SV Default	X-Prop	X-Opt
X	0	0	0	X	0
X	0	1	1	X	X
X	1	0	0	X	X
X	1	1	1	X	1

Find your X with SV Assertions

- Using immediate assertions is a simple method to monitor for X in a design.
- They can be applied within a functional module.
  - To monitor combinational if/else, case, and continuous assign statements.
- They can be applied at the test bench level.
  - For ports and flip-flop outputs, this method should be considered, since it could be applied to the design both before and after synthesis.
- They can be disabled
- Severity levels available for reporting
- Automatically ignored by synthesis

System Verilog Immediate Assertions Syntax

Don Mills
MICROCHIP

- For syntax for SystemVerilog immediate assertion

assert (expession) pass_statement(s) [**else** fail_statment(s)]

- SystemVerilog built-in severity levels
  - $fatal - ends the simulation
  - $error - gives a runtime error, but simulation continues
  - $warning - gives a runtime warning, simulation continues
  - $info - prints the specified messag

SystemVerilog Assertions if/else statement

Don Mills
MICROCHIP

- Simple immediate expression using default error statement

```
always_comb begin
  assert (!$isunknown(sel));
  if (sel == 1'b1)  out = a;
  else              out = b;
end
```

- Example with error message included

```
always_comb begin
  assert (!$isunknown(sel))
    else $error("%m, sel = X");
  if (sel == 1'b1)  out = a;
  else              out = b;
end
```

- Could use a 'define macro for the assertion code or SV "let" statement

Use macro's with arguments

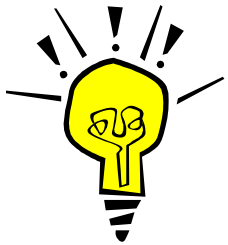
- Replace the immediate assertion with a `define macro

```
`define assert_X(arg) assert (!$isunknown(arg));

always_comb begin
  `assert_X(sel)
  if (sel == 1'b1)  out = a;
  else              out = b;
end
```

```
`define assert_X(arg, msg="")  \
  assert (!$isunknown(arg))    \
  else $error("%m,%s", $time, msg);

always_comb begin
  `assert_X(sel,"sel=X")
  if (sel == 1'b1)  out = a;
  else              out = b;
end
```



SystemVerilog Assertions case statement

Don Mills
MICROCHIP

```
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
```

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

SystemVerilog Assertions continuous assignments

Don Mills
MICROCHIP

```
assign out_or   = a | b;
assign out_and  = c & d;
assign out_tern = sel ? e : f;
assign x_check  = {a,b,c,d,e,f,sel};

always_comb begin
  assert (!$isunknown(x_check))
    else $error("%m, comb_logic = X");
end
```

OR

```
always_comb begin
  `assert_X(x_check,"comb_logic=X")
end
```

SystemVerilog Assertions for GLS as well as RTL

Don Mills
MICROCHIP

- Ports and flip-flop output names are preserved through synthesis
  - —*At least a some variation of a flip-flop output name is preserved.*
- Recommendation
  - —Put assertion X checks for ports and FF's in a side file that is then bound to the design

SystemVerilog Assertions Port check example

Don Mills

MICROCHIP

```
module foo(input  logic in1, in2, din, clk,
           output logic out1, out2);

  assign out1 = in1 & in2;
  always_ff@(posedge clk)
     out2 <= din;
endmodule
```

```
module PortCheck(input logic in1, in2, din, clk, out1, out2);

logic [5:0] sig_list
assign sig_list = {in1, in2, din, clk, out1, out2}
 `assert_X(sig_list),”X in foo ports”)
endmodule
```

SystemVerilog Assertions Port check example Cont'

Don Mills
MICROCHIP

- Bind is like a local instantiation

```
module top;
  logic in1, in2 ...
  foo f(.*)
  test tb(.*)
  bind foo PortCheck X_ports (.*);
  ...
endmodule
```

Module PortCheck (instance X_ports) will be pseudo-instantiated into module foo

Disabling assertions

- X checking for assertions are not always welcome
  - —During boot-up
  - —UPF low power mode
- SystemVerilog Assertions built in system tasks disable/enable assertions
  - —$assertoff and $asserton
- Arguments to $assertoff and $asserton allow the user to specify
  - — A list of assertions
  - — A list of modules
  - — A hierarchy depth (levels down the hierarchy)

Conclusions & Recommendations

Don Mills
MICROCHIP

- SystemVerilog assertions for X detection
  - easy to use
  - built into the language
  - ignored by synthesis
  - can be turned on and off real time during simulations.
- Use SystemVerilog assertions to monitor for X's
  - check all conditional tests, IE. if (sel) or case expression
  - check all inputs and/or outputs of module
- Assertion checks are better than tool based X-propagation hooks
  - they tell where the problem starts
  - (could [illegible]s)

BUT WAIT, There's more:
Mentors best kept

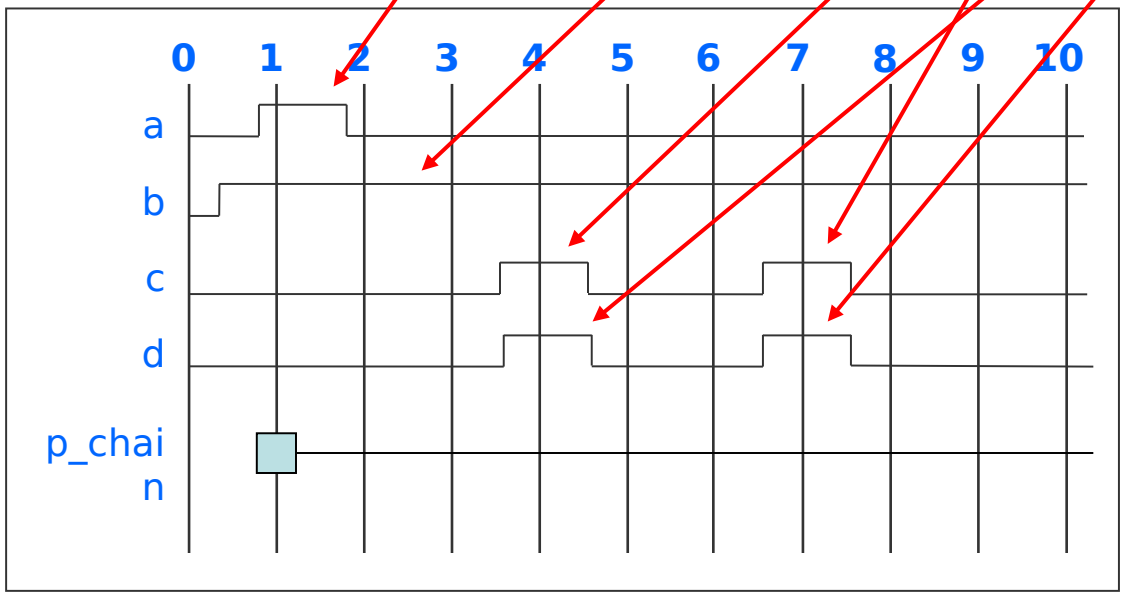
MENTOR’S BEST KEPT SECRET

Concurrent Assertion with chained implications

Don Mills
MICROCHIP

```
property p_chain;
  @(posedge clk) $rose(a) |->  b[*0:$] ##1 c  |->  d;
endproperty:p_chain;

ap_chain: assert property (p_chain);
```

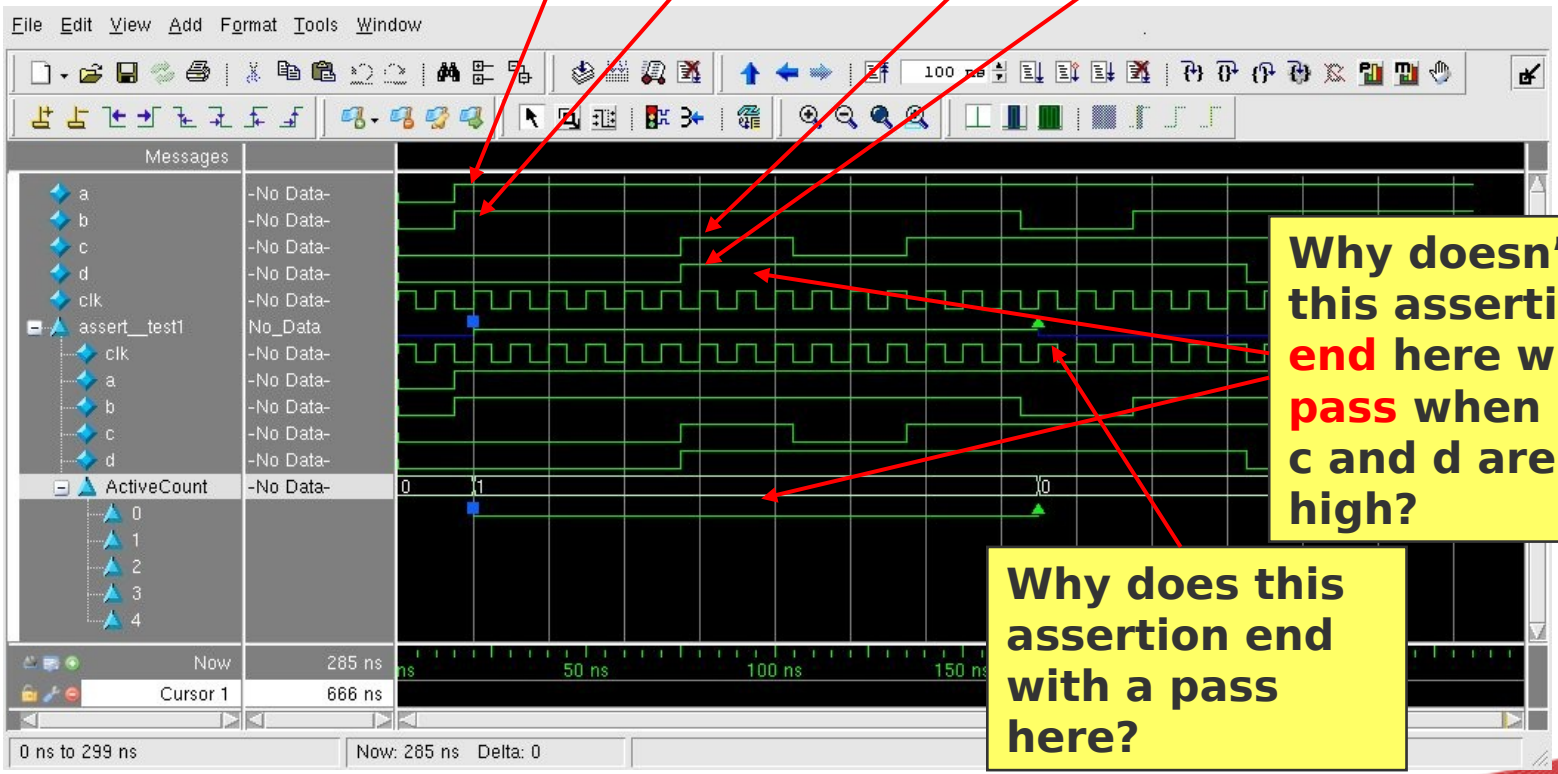


Does not end with a pass at cycle 4 or cycle 7

Another look…

```
property p_chain;
  @(posedge clk) $rose(a) |->  b[*0:$] ##1 c  |->  d;
endproperty:p_chain;

ap_chain: assert property (p_chain);
```



Why doesn't this assertion end here with a pass when both c and d are high?

Why does this assertion end with a pass here?

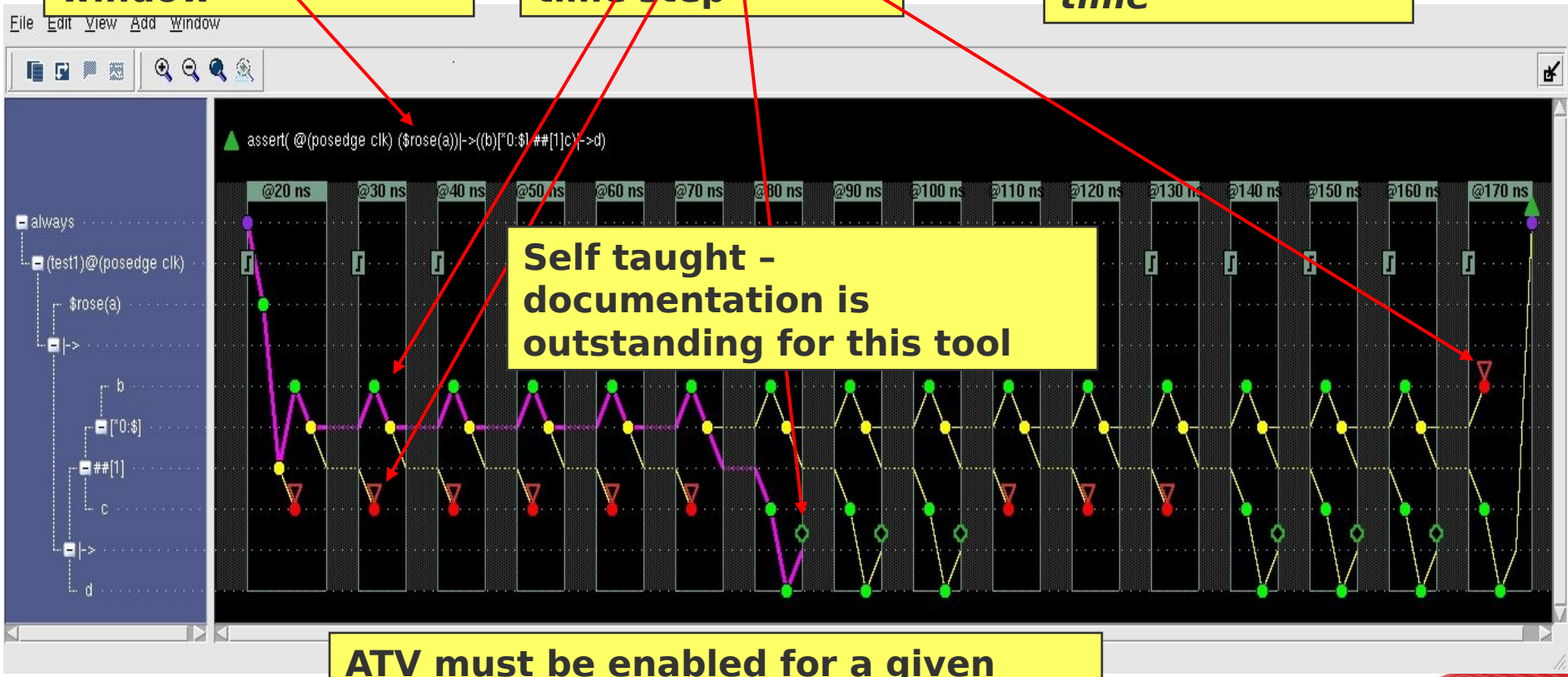
Don Mills

Assertion Thread Viewer (ATV)

Assertion statement is listed at the top of the window

Pass/Fail for tested signals is noted for each time step

Only one Assertion thread is listed at a time



Self taught – documentation is outstanding for this tool

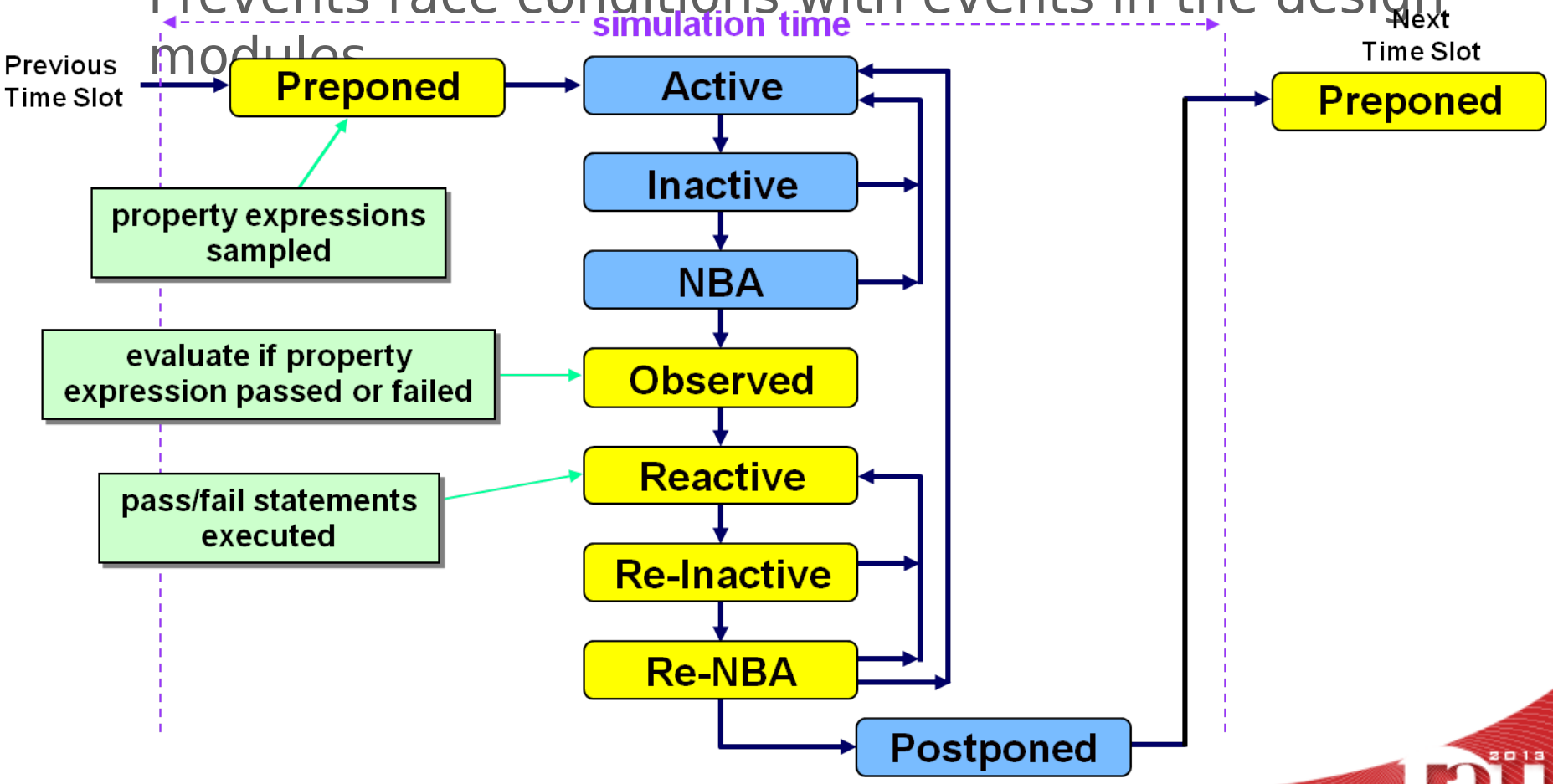
ATV must be enabled for a given assertion and a specific thread of that assertion prior to running a simulation

Concurrent Assertions Use Special Event Scheduling

Don Mills

MICROCHIP

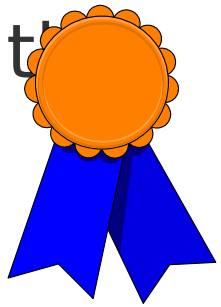
- Concurrent assertions use special event scheduling queues
  - Prevents race conditions with events in the design modules



(Mentor's Best Kept Secret)

Don Mills
MICROCHIP

- Concurrent Assertions are sampled at t
  beginning of a time step



  - If data is changing during the time step
    - Must look at the data prior to the time step
    - Like a D-FF – must look at the value of D prior the clk edge
  - ATV doesn't show the value, it shows the pass/fail for each time step


2013
U2U
USER2USER