ENTER THE TERRIFYING WORLD OF SIMULATION / SYNTHESIS LOGIC MISMATCHES

GRAPHCORE

v0.1.3

Will these SV simulators give the same output?

Compile, elab & sim with default settings

```
> sim_a dut.sv testbench.sv

> sim_b dut.sv testbench.sv
```

Will these SV simulators give the same output?

NO

Compile, elab & sim with default settings

```
> sim_a dut.sv testbench.sv
Error: "testbench.sv", 6: testbench: at time 0
Test failed

> sim_b dut.sv testbench.sv
Test passed
```

```
> cat dut.sv
`define FAIL_IF_DEFINED
```

```
> cat testbench.sv
module testbench;
  `ifdef FAIL_IF_DEFINED
    initial $error("Test failed");
  `else
    initial $display("Test passed");
  `endif
endmodule
```

COMPILATION UNITS

SystemVerilog Language Reference Manual

SV LRM (1800-2017) 3.12.1 Compilation units

— **compilation unit:** *A* collection of one or more SystemVerilog source files compiled together.

COMPILATION UNITS

SV LRM (1800-2017) 3.12.1 Compilation units

The exact mechanism for defining which files constitute a compilation unit is **tool-specific**.
... compliant tools shall [support]:

a) All files on a given compilation command line make a single compilation unit ...

b) Each file is a separate compilation unit ...

Visibility of `` `define `` change design logic

Sim/Synth differences invalidate verification

What is a design delivery?

- RTL
- How do you tell the back end ‘customers’ which RTL files to compile in a compilation unit?
- How do you know they actually do it?

GLS TO THE RESCUE

GLS TO THE RESCUE

Gate Level Simulations (GLS)

Simulate a post synthesis netlist (gates) c.f. RTL

Main opportunity to check RTL processed correctly by our ‘customers’

GLS SPEEDUP TECHNIQUES

1. Block level
2. Synth to primitives (not timing clean netlist)
   - Quicker to synth & quicker to sim
   - Also avoids most X-pessimism issues

$ERROR TRAPS

Lay $error elab ‘traps’ in RTL to catch incorrect compilation units?

COMMAND LINE DEFINES

- Eliminate command line `` `define ``s from sim and synth tools
  - Use SV `` `define `` macros instead
- Audit & understand all `` `define `` macros

GENUINE RTL BUG UNDETECTED BY VERIF SIMS

```
always_ff @(posedge clk)
  missile_detected_reg <= missile_detected_ip;

always_comb
  if (missile_detected_reg)
    launch_intercept_op = 1;
  else
    launch_intercept_op = 0;
```

What is the RTL Bug?

```
always_ff @(posedge clk)
  missile_detected_reg <= missile_detected_ip;

always_comb
  if (missile_detected_reg)
    launch_intercept_op = 1;
  else
    launch_intercept_op = 0;
```

Why can’t simulations detect bug?

```
always_ff @(posedge clk)
  missile_detected_reg <= missile_detected_ip;

always_comb
  if (missile_detected_reg)
    launch_intercept_op = 1;
  else
    launch_intercept_op = 0;
```

<< This condition 'optimistically' cleans 'X' to '0'

'X's can be '1' In Real Life of course

Not simulating logic that will be synthesized

RTL Sim

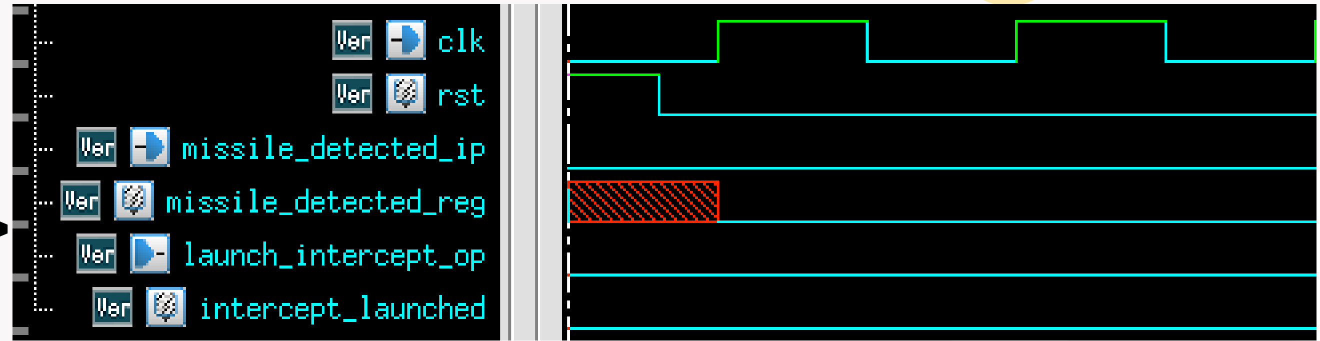
```
> run_rtl_sim
Pass:  intercept not launched
```

Gate Level Sim

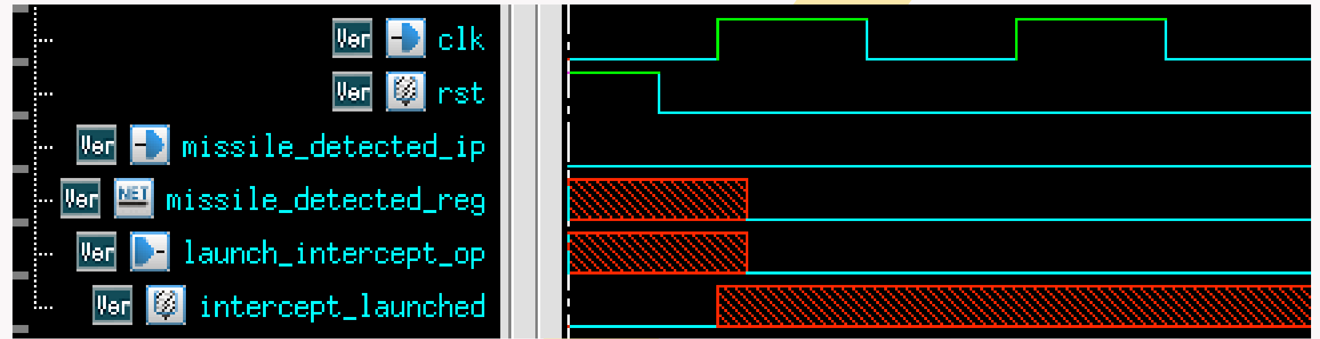
```
> run_gls
Error: intercept launched
```

RTL Sim

Cleans up X ☹ >>

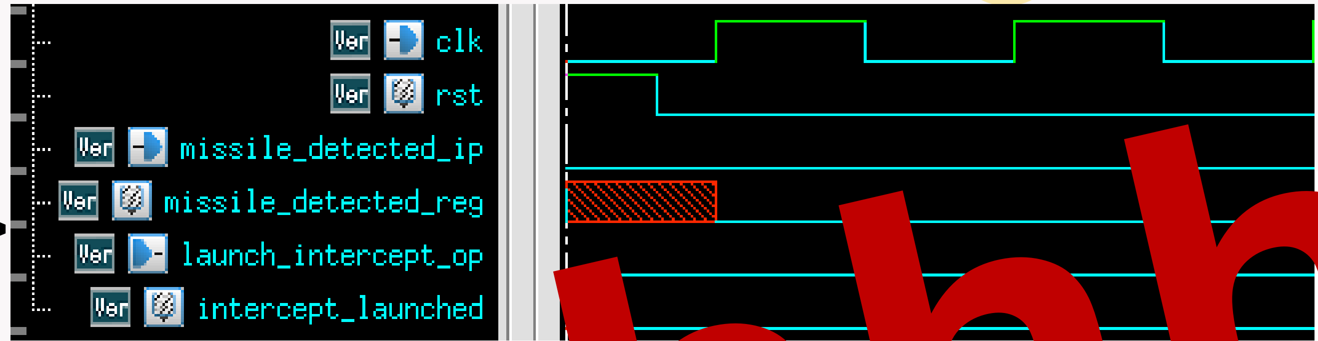


Gate Level Sim



RTL Sim

Cleans up X ☹ >>



Gate Level Sim

Aaagghhh

!!!

WHY IS THIS HAPPENING?

SystemVerilog LRM is the problem

Sims that follow LRM don’t match synth logic

Eeek!

Known as ‘X Optimism’

SOLUTION

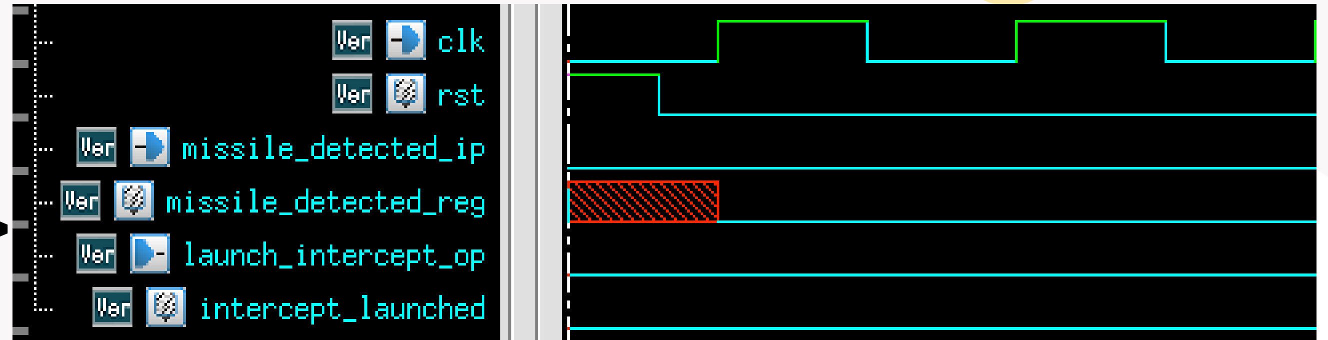
Simulators have a ‘realistic x propagation’ mode

Aim: Propagate Xs like real hardware

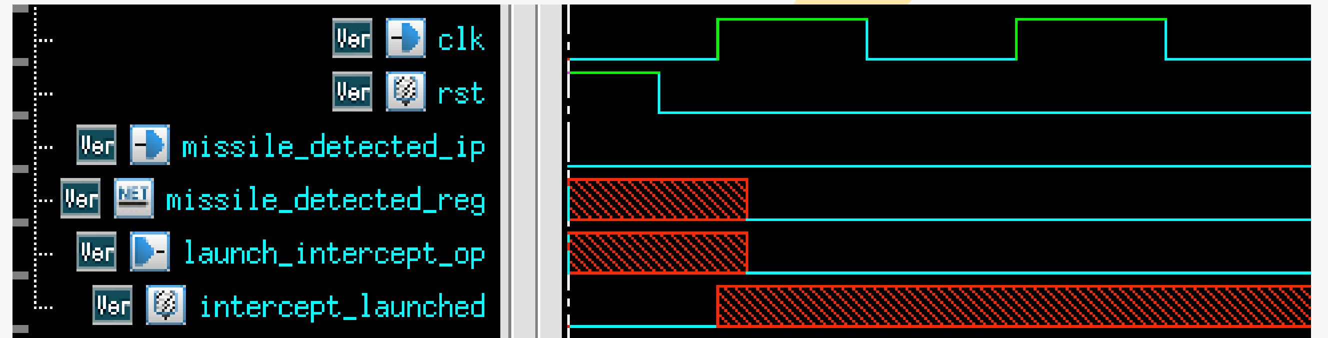
Abandon following LRM

RTL Sim

Cleans up X ☹ >>

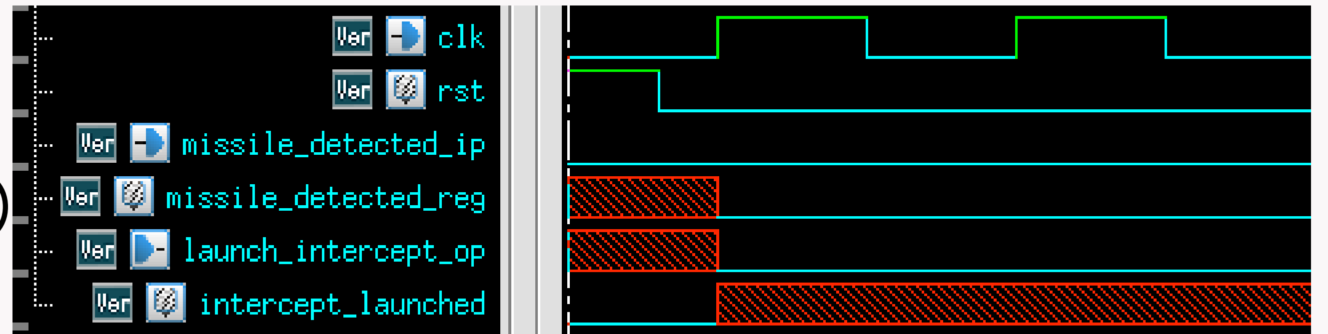


Gate Level Sim



RTL Xprop mode

Matches hardware ☺



Realistic x-prop modes differ on all simulators

Behaviour not defined by SV LRM (obviously)

Realistic RTL x-prop modes ‘quite’ effective

Detect ~95% bugs (my experience)

Run GLS to detect 100% X optimism bugs ☺

UNIQUE CASES

RTL SYNTH

```systemverilog
logic [31:0] a,b,c,d;
logic        e,z;

always_comb
  case (1'b1)
    a == 5       : z = 0;
    b < 24000    : z = 0;
    c == 2**d    : z = 0;
    e == 1       : z = 1;
  endcase
```

```
GTECH_LD1 z_reg ( .G(N39), .D(N40), .Q(z) );
GTECH_AND4 U52 ( .A(e), .B(n54), .C(n55), .D(n56), .Z(N40) );
GTECH_NAND4 U53 ( .A(n54), .B(n55), .C(n56), .D(n57), .Z(N39) );
GTECH_NOT U54 ( .A(e), .Z(n57) );
GTECH_OR8 U55 ( .A(b[24]), .B(b[23]), .C(b[26]), .D(b[25]), .E(b[28]), .F(
      b[27]), .G(n58), .H(n59), .Z(n56) );
GTECH_OR8 U56 ( .A(b[22]), .B(b[21]), .C(b[20]), .D(b[19]), .E(b[18]), .F(
      b[17]), .G(b[16]), .H(n60), .Z(n59) );
GTECH_AO21 U57 ( .A(n61), .B(b[14]), .C(b[15]), .Z(n60) );
GTECH_OA21 U58 ( .A(n62), .B(b[13]), .C(n63), .Z(n61) );
GTECH_OR3 U59 ( .A(b[9]), .B(b[13]), .C(n64), .Z(n63) );
GTECH_AND3 U60 ( .A(b[7]), .B(b[6]), .C(b[8]), .Z(n64) );
GTECH_AND3 U61 ( .A(b[11]), .B(b[10]), .C(b[12]), .Z(n62) );
GTECH_OR3 U62 ( .A(b[31]), .B(b[30]), .C(b[29]), .Z(n58) );
GTECH_NAND3 U63 ( .A(n65), .B(n66), .C(n67), .Z(n55) );
GTECH_AND5 U64 ( .A(n68), .B(n69), .C(n70), .D(n71), .E(n72), .Z(n67) );
GTECH_MUXI2 U65 ( .A(n73), .B(n74), .S(n75), .Z(n72) );
GTECH_AND3 U66 ( .A(d[3]), .B(n76), .C(d[4]), .Z(n75) );
GTECH_NAND4 U67 ( .A(n77), .B(n78), .C(n79), .D(n80), .Z(n74) );
GTECH_OA22 U68 ( .A(c[30]), .B(n81), .C(c[31]), .D(n82), .Z(n80) );
GTECH_OA22 U69 ( .A(c[28]), .B(n83), .C(c[29]), .D(n84), .Z(n79) );
GTECH_OA22 U70 ( .A(c[26]), .B(n85), .C(c[27]), .D(n86), .Z(n78) );
GTECH_OA22 U71 ( .A(c[25]), .B(n87), .C(c[24]), .D(n88), .Z(n77) );
GTECH_NOT U72 ( .A(n89), .Z(n87) );
GTECH_OR8 U73 ( .A(c[24]), .B(c[25]), .C(c[26]), .D(c[27]), .E(c[28]), .F(
      c[29]), .G(c[30]), .H(c[31]), .Z(n73) );
GTECH_OA22 U74 ( .A(n89), .B(n90), .C(n91), .D(n92), .Z(n71) );
GTECH_NOR4 U75 ( .A(c[0]), .B(c[16]), .C(c[24]), .D(c[8]), .Z(n92) );
GTECH_NOR4 U76 ( .A(c[17]), .B(c[1]), .C(c[25]), .D(c[9]), .Z(n90) );
GTECH_MUXI2 U77 ( .A(n93), .B(n94), .S(n95), .Z(n70) );
GTECH_AND3 U78 ( .A(n76), .B(n96), .C(d[3]), .Z(n95) );
GTECH_NAND4 U79 ( .A(n97), .B(n98), .C(n99), .D(n100), .Z(n94) );
GTECH_OA22 U80 ( .A(c[14]), .B(n81), .C(c[15]), .D(n82), .Z(n100) );
GTECH_OA22 U81 ( .A(c[12]), .B(n83), .C(c[13]), .D(n84), .Z(n99) );
GTECH_OA22 U82 ( .A(c[10]), .B(n85), .C(c[11]), .D(n86), .Z(n98) );
GTECH_AOI2N2 U83 ( .A(n101), .B(n89), .C(c[8]), .D(n88), .Z(n97) );
GTECH_NOT U84 ( .A(c[9]), .Z(n101) );
GTECH_OR8 U85 ( .A(c[10]), .B(c[11]), .C(c[12]), .D(c[13]), .E(c[14]), .F(
      c[15]), .G(c[8]), .H(c[9]), .Z(n93) );
GTECH_MUXI2 U86 ( .A(n102), .B(n103), .S(n104), .Z(n69) );
GTECH_AND3 U87 ( .A(n105), .B(n96), .C(n76), .Z(n104) );
GTECH_NOT U88 ( .A(d[4]), .Z(n96) );
GTECH_NAND4 U89 ( .A(n106), .B(n107), .C(n108), .D(n109), .Z(n103) );
GTECH_OA22 U90 ( .A(c[6]), .B(n81), .C(c[7]), .D(n82), .Z(n109) );
GTECH_OA22 U91 ( .A(c[4]), .B(n83), .C(c[5]), .D(n84), .Z(n108) );
GTECH_OA22 U92 ( .A(c[2]), .B(n85), .C(c[3]), .D(n86), .Z(n107) );
GTECH_AOI2N2 U93 ( .A(n110), .B(n89), .C(c[0]), .D(n88), .Z(n106) );
GTECH_NOT U94 ( .A(c[1]), .Z(n110) );
GTECH_OR8 U95 ( .A(c[0]), .B(c[1]), .C(c[2]), .D(c[3]), .E(c[4]), .F(c[5]),
      .G(c[6]), .H(c[7]), .Z(n102) );
GTECH_MUXI2 U96 ( .A(n111), .B(n112), .S(n113), .Z(n68) );
GTECH_AND3 U97 ( .A(n76), .B(n105), .C(d[4]), .Z(n113) );
GTECH_NOT U98 ( .A(d[3]), .Z(n105) );
GTECH_NOR8 U99 ( .A(d[12]), .B(d[11]), .C(d[10]), .D(d[15]), .E(d[14]), .F(
      d[13]), .G(n114), .H(n115), .Z(n76) );
GTECH_OR8 U100 ( .A(d[25]), .B(d[24]), .C(d[23]), .D(n116), .E(d[5]), .F(
      d[31]), .G(d[30]), .H(n117), .Z(n115) );
GTECH_OR4 U101 ( .A(d[7]), .B(d[6]), .C(d[9]), .D(d[8]), .Z(n117) );
GTECH_OR4 U102 ( .A(d[27]), .B(d[26]), .C(d[29]), .D(d[28]), .Z(n116) );
GTECH_OR4 U103 ( .A(d[18]), .B(d[17]), .C(d[16]), .D(n118), .Z(n114) );
GTECH_OR4 U104 ( .A(d[20]), .B(d[19]), .C(d[22]), .D(d[21]), .Z(n118) );
GTECH_NAND4 U105 ( .A(n119), .B(n120), .C(n121), .D(n122), .Z(n112) );
GTECH_OA22 U106 ( .A(c[22]), .B(n81), .C(c[23]), .D(n82), .Z(n122) );
GTECH_OA22 U107 ( .A(c[20]), .B(n83), .C(c[21]), .D(n84), .Z(n121) );
GTECH_OA22 U108 ( .A(c[18]), .B(n85), .C(c[19]), .D(n86), .Z(n120) );
GTECH_AOI2N2 U109 ( .A(n123), .B(n89), .C(c[16]), .D(n88), .Z(n119) );
GTECH_NOT U110 ( .A(n91), .Z(n88) );
GTECH_NOR3 U111 ( .A(d[1]), .B(d[2]), .C(d[0]), .Z(n91) );
GTECH_NOR3 U112 ( .A(d[1]), .B(d[2]), .C(n124), .Z(n89) );
GTECH_NOT U113 ( .A(d[0]), .Z(n124) );
GTECH_NOT U114 ( .A(c[17]), .Z(n123) );
GTECH_OR8 U115 ( .A(c[16]), .B(c[17]), .C(c[18]), .D(c[19]), .E(c[20]), .F(
      c[21]), .G(c[22]), .H(c[23]), .Z(n111) );
GTECH_AOI222 U116 ( .A(n125), .B(n84), .C(n126), .D(n82), .E(n127), .F(n81),
      .Z(n66) );
GTECH_OR3 U117 ( .A(n128), .B(d[0]), .C(n129), .Z(n81) );
GTECH_OR4 U118 ( .A(c[14]), .B(c[22]), .C(c[30]), .D(c[6]), .Z(n127) );
GTECH_NAND3 U119 ( .A(d[1]), .B(d[0]), .C(d[2]), .Z(n82) );
GTECH_OR4 U120 ( .A(c[15]), .B(c[23]), .C(c[31]), .D(c[7]), .Z(n126) );
GTECH_NAND3 U121 ( .A(d[0]), .B(n128), .C(d[2]), .Z(n84) );
GTECH_OR4 U122 ( .A(c[13]), .B(c[21]), .C(c[29]), .D(c[5]), .Z(n125) );
GTECH_AOI222 U123 ( .A(n130), .B(n85), .C(n131), .D(n83), .E(n132), .F(n86),
      .Z(n65) );
GTECH_NAND3 U124 ( .A(d[0]), .B(n129), .C(d[1]), .Z(n86) );
GTECH_OR4 U125 ( .A(c[11]), .B(c[19]), .C(c[27]), .D(c[3]), .Z(n132) );
GTECH_OR3 U126 ( .A(d[0]), .B(d[1]), .C(n129), .Z(n83) );
GTECH_NOT U127 ( .A(d[2]), .Z(n129) );
GTECH_OR4 U128 ( .A(c[12]), .B(c[20]), .C(c[28]), .D(c[4]), .Z(n131) );
GTECH_OR3 U129 ( .A(d[0]), .B(d[2]), .C(n128), .Z(n85) );
GTECH_NOT U130 ( .A(d[1]), .Z(n128) );
GTECH_OR4 U131 ( .A(c[10]), .B(c[18]), .C(c[26]), .D(c[2]), .Z(n130) );
GTECH_NAND3 U132 ( .A(n133), .B(n134), .C(n135), .Z(n54) );
GTECH_NOR8 U133 ( .A(n136), .B(n137), .C(a[11]), .D(a[10]), .E(a[15]), .F(
      a[14]), .G(a[13]), .H(a[12]), .Z(n135) );
GTECH_NAND2 U134 ( .A(a[2]), .B(a[0]), .Z(n137) );
GTECH_OR8 U135 ( .A(a[17]), .B(a[16]), .C(a[19]), .D(a[18]), .E(a[20]), .F(
      a[1]), .G(a[22]), .H(a[21]), .Z(n136) );
GTECH_NOR8 U136 ( .A(a[9]), .B(a[8]), .C(a[7]), .D(a[6]), .E(a[5]), .F(a[4]),
      .G(a[3]), .H(a[31]), .Z(n134) );
GTECH_NOR8 U137 ( .A(a[30]), .B(a[29]), .C(a[28]), .D(a[27]), .E(a[26]), .F(
      a[25]), .G(a[24]), .H(a[23]), .Z(n133) );
```

What does this RTL synthesize to?

```
always_comb
  unique case (1'b1)
    a == 5      : z = 0;
    b < 24000   : z = 0;
    c == 2**d   : z = 0;
    e == 1      : z = 1;
  endcase
```

What does this RTL synthesize to?

```
always_comb
  unique case (1'b1)
    a == 5       : z = 0;
    b < 24000    : z = 0;
    c == 2**d    : z = 0;
    e == 1       : z = 1;
  endcase
```

```
assign z = e;
```

No logic at all.
It is just a wire.

Why?

What does this RTL synthesize to?

```
always_comb
  unique case (1'b1)
    a == 5      : z = 0;
    b < 24000   : z = 0;
    c == 2**d   : z = 0;
    e == 1      : z = 1;
  endcase
```

```
assign z = e;
```

No logic at all.
It is just a wire.

Why?

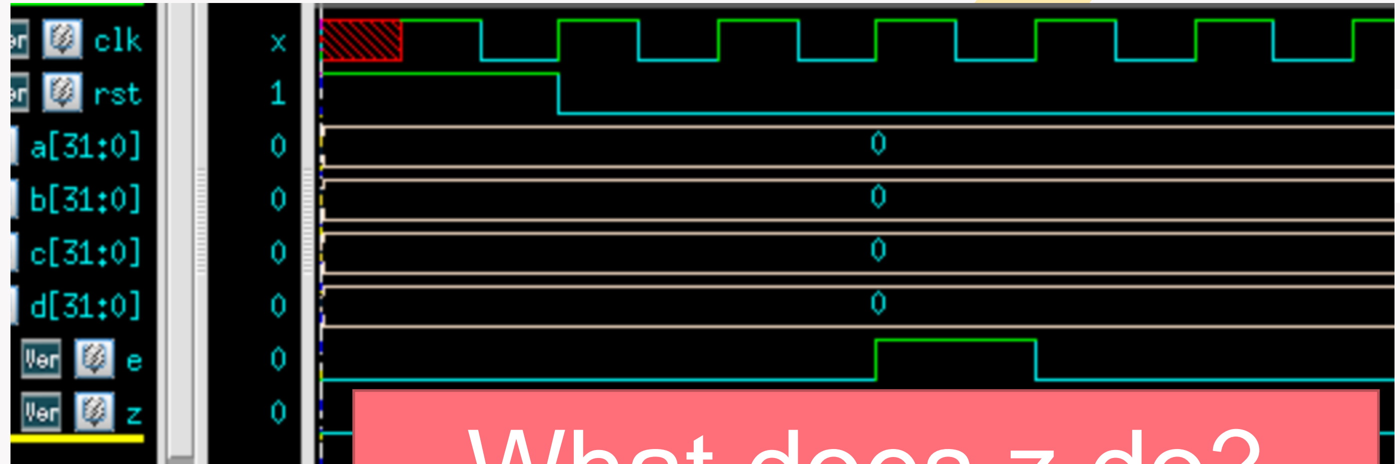
e and z == 0 for all other cases

Unique cases:

- really powerful
  - Too powerful?
- Area reduction
- Performance increase
- But ...

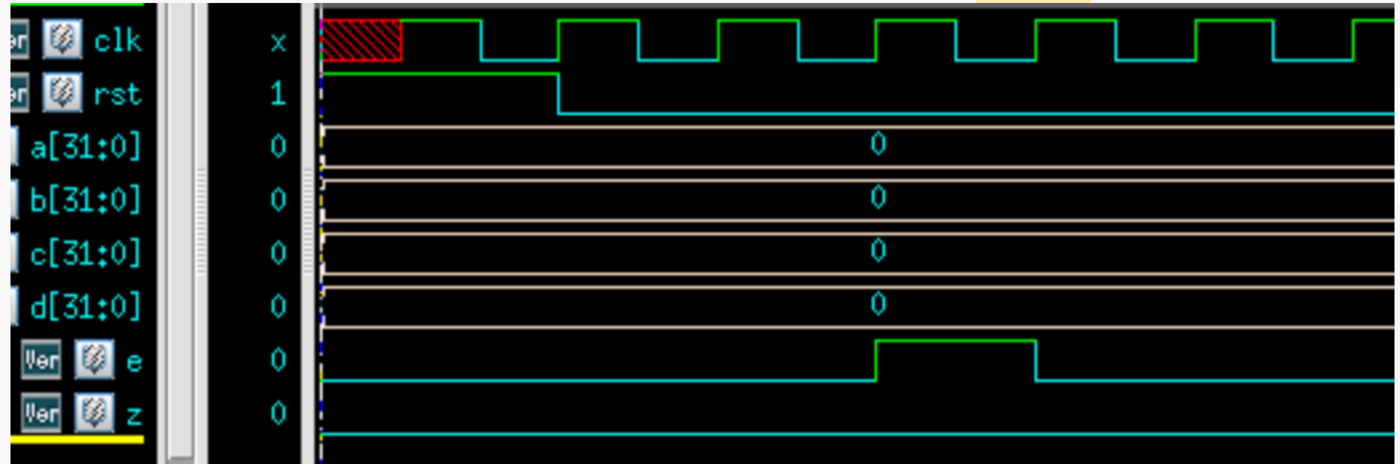
Synth: z is driven by a wire from e

```
always_comb
  unique case (1'b1)
    a == 5       : z = 0;
    b < 24000    : z = 0;
    c == 2**d    : z = 0;
    e == 1       : z = 1;
  endcase
```



Synth: z is driven by a wire from e

```
always_comb
  unique case (1'b1)
    a == 5      : z = 0;
    b < 24000   : z = 0;
    c == 2**d   : z = 0;
    e == 1      : z = 1;
  endcase
```



Synth: z is driven by a wire from e

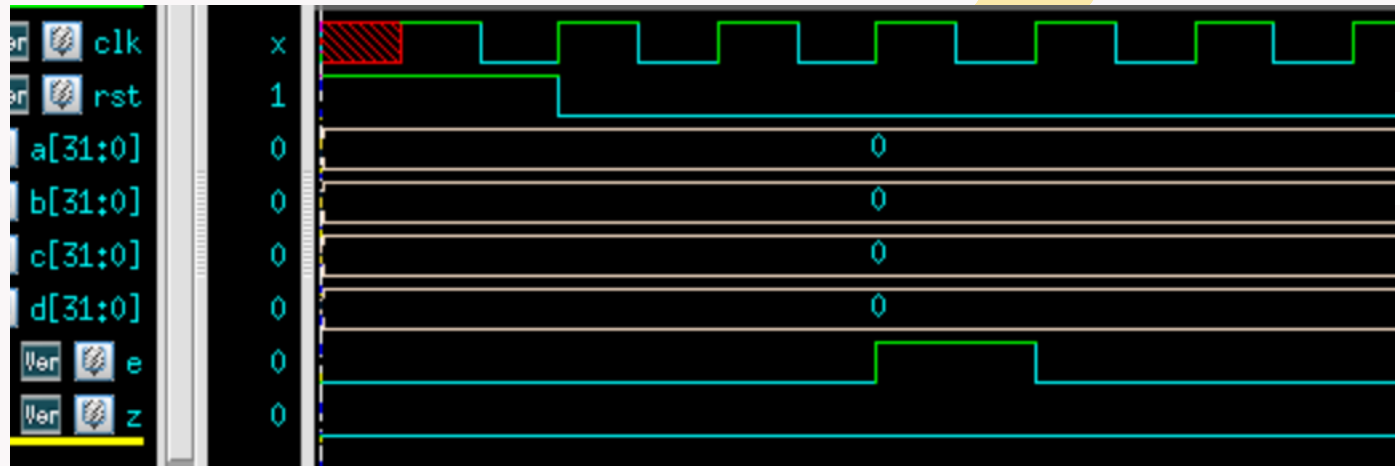
```
always_comb
  unique case (1'b1)
    a == 5      : z = 0;
    b < 24000   : z = 0;
    c == 2**d   : z = 0;
    e == 1      : z = 1;
  endcase
```

clk x
rst 1
a[31:0] 0 0
b[31:0] 0 0
c[31:0] 0 0
d[31:0] 0 0
e 0
z 0

?@

Synth: z is driven by a wire from e

```
always_comb
  unique case (1'b1)
    a == 5       : z = 0;
    b < 24000    : z = 0;
    c == 2**d    : z = 0;
    e == 1       : z = 1;
  endcase
```




```
Warn(UC-7) Multiple matching unique case conditions
xm_unique_case.sv (14) at 3500ps
```

Synth: z is driven by a wire from e

```
always_comb
  unique case (1'b1)
    a == 5       : z = 0;
    b < 24000    : z = 0;
    c [illegible] : z = 0;
    e == [illegible] : z = 1;
  endcase
```

Warn([illegible]) Multiple matching unique case conditions
xm_unique_case.sv([illegible]14) at 3500ps

This warning means the verification is invalid

Don't miss it

UNIQUE CASE CHECKLIST

1. Promote case qualifier warnings to errors
2. Monitor $assertioncontrol usage

```
$assertcontrol(ASSERTIONS_OFF, UNIQUE|UNIQUE0|PRIORITY);
```

3. Formal Tools: run auto properties
4. GLS regressions
5. Discuss with colleagues: ban unique cases?

Anthony Wood
anthonyw@graphcore.ai