Finding and Understanding Bugs in FPGA Synthesis Tools

Verismith: FPGA Synthesis Tool Fuzzer

Yann Herklotz, John Wickerson

February 26, 2020

Imperial College London

Why find bugs?

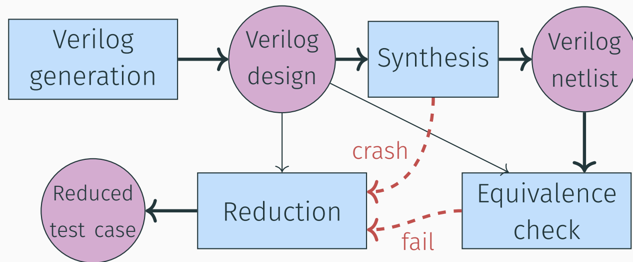
- Designers have to trust the synthesis tool to do the right job
- Bugs that generate wrong code can be hard to debug
- Bugs that crash the tool can affect tool flows and be frustrating

Why find bugs?

- Designers have to trust the synthesis tool to do the right job
- Bugs that generate wrong code can be hard to debug
- Bugs that crash the tool can affect tool flows and be frustrating

- Use **Verismith** to improve reliability of synthesis tools

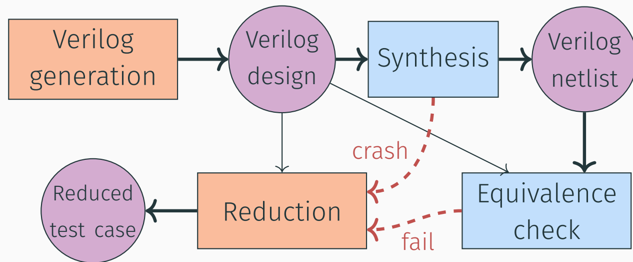
Verismith



Main contributions

- Synthesis tool fuzzing framework

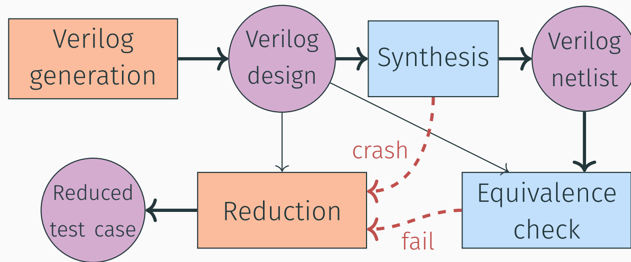
Verismith



Main contributions

- Synthesis tool fuzzing framework
- Behavioural and deterministic Verilog generation
- Efficient Verilog Reduction

Verismith



Main contributions

- Synthesis tool fuzzing framework
- Behavioural and deterministic Verilog generation
- Efficient Verilog Reduction

Synthesis tools tested

Quartus
Vivado
XST
Yosys

What is deterministic Verilog?

- Only one interpretation of the design
- Nondeterminism example:
  Any undefined values can be 1 or 0

What is deterministic Verilog?

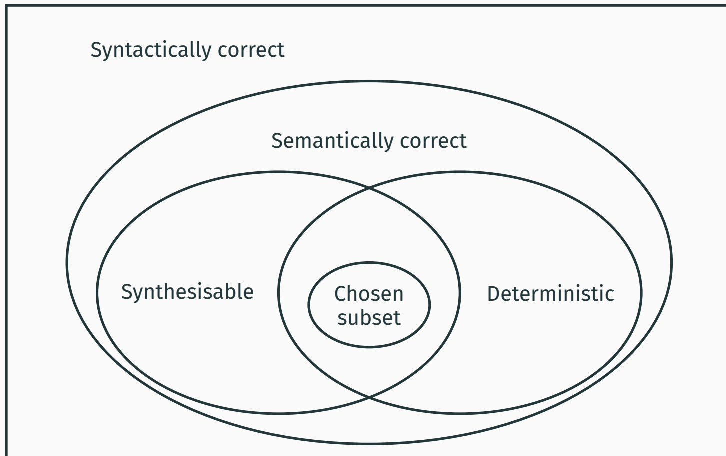
- Only one interpretation of the design
- Nondeterminism example:
  Any undefined values can be 1 or 0

What is a bug?

- Synthesis tool crashes
- Synthesis tool outputs the wrong netlist

Verilog 2005 standards

- Verilog for simulation
- Synthesisable Verilog



Nondeterministic simulation example

```
always @(posedge clk)
    a = b;

always @(posedge clk)
    b = c;
```

- Simulation will run the always blocks in any order
- This will synthesise correctly
- We therefore get a mismatch between synthesis and simulation

Nondeterministic simulation example

```
always @(posedge clk)
    a <= b;

always @(posedge clk)
    b <= c;
```

- Simulation will run the always blocks in any order
- This will synthesise correctly
- We therefore get a mismatch between synthesis and simulation
- Adding nonblocking assignment in sequential always blocks fixes this

```
module top (output y, input [2:0] w);
  assign y = 1'b1 >> (w * (3'b110));
endmodule
```

- Bug in a development version of Yosys[1]
- Result not truncated properly, which results in an unwanted shift

[1] https://github.com/YosysHQ/yosys/issues/1047

Motivating Bug: Yosys

```
module top (output y, input [2:0] w);
  assign y = 1'b1 >> (3'b100 * (3'b110));
endmodule
```

- Bug in a development version of Yosys[1]
- Result not truncated properly, which results in an unwanted shift

[1] https://github.com/YosysHQ/yosys/issues/1047

Motivating Bug: Yosys

```
module top (output y, input [2:0] w);
  assign y = 1'b1 >> 6'b110000;
endmodule
```

- Bug in a development version of Yosys[1]
- Result not truncated properly, which results in an unwanted shift

[1] https://github.com/YosysHQ/yosys/issues/1047

Motivating Bug: Yosys

```
module top (output y, input [2:0] w);
  assign y = 1'b0;
endmodule
```

- Bug in a development version of Yosys[1]
- Result not truncated properly, which results in an unwanted shift

[1] https://github.com/YosysHQ/yosys/issues/1047

Motivating Bug: Yosys

```
module top (output y, input [2:0] w);
  assign y = 1'b1 >> 3'b000;
endmodule
```

- Bug in a development version of Yosys[1]
- Result not truncated properly, which results in an unwanted shift

[1] https://github.com/YosysHQ/yosys/issues/1047

Motivating Bug: Yosys

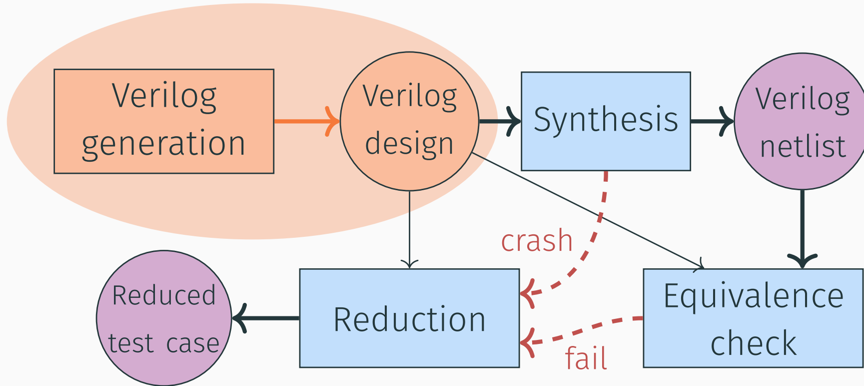
```
module top (output y, input [2:0] w);
  assign y = 1'b1;
endmodule
```

- Bug in a development version of Yosys[1]
- Result not truncated properly, which results in an unwanted shift

[1] https://github.com/YosysHQ/yosys/issues/1047

Example Verismith Run

Run Outline: Verilog Generation



Verilog generation

```verilog
// -*- mode: verilog -*-
module top #(parameter param30 = (8'hbb)) (y, clk, wire0, wire1, wire2, wire3);
  output [(32'hb7):(32'h0)] y;
  input [(1'h0):(1'h0)] clk;
  input signed [(5'h11):(1'h0)] wire0;
  input signed [(4'ha):(1'h0)] wire1;
  input [(4'hd):(1'h0)] wire2;
  input [(4'h8):(1'h0)] wire3;
  wire signed [(4'hb):(1'h0)] wire27;
  wire [(5'h15):(1'h0)] wire26;
  wire [(5'h10):(1'h0)] wire25;
  wire [(5'h13):(1'h0)] wire24;
  reg signed [(4'he):(1'h0)] reg4 = (1'h0);
  reg [(2'h3):(1'h0)] reg5 = (1'h0);
  reg [(5'h14):(1'h0)] reg6 = (1'h0);
  reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
  reg [(4'hd):(1'h0)] reg8 = (1'h0);
  wire [(4'hd):(1'h0)] wire9;
  wire [(4'he):(1'h0)] wire10;
  wire signed [(2'h2):(1'h0)] wire22;
  assign y = {wire27, wire26, wire25, wire24, reg4,
              reg5, reg6, reg7, reg8, wire9, wire10, wire22};
  always
    @(posedge clk) begin
      reg4 <=  wire1;
      if ($unsigned((~&(8'hb2))))
        begin
          reg5 <=  reg4;
          reg6 <=  wire1;
        end
      else
        begin
          reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
          reg6 <=  reg6;
        end
    end
  always @* begin
     reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 != (8'h9d))))) <<< ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                wire3 : $signed(wire1))) ? $unsigned(((^wire0) + $unsigned(wire3))) : (((reg5 * wire3) ?
                    wire1 : $unsigned(reg6)) ? {{reg4, wire2}} : (reg5[(1'h0):(1'h0)] ? $signed(reg4) : (~wire3)))));
     reg8 = (~^$unsigned(reg6));
  end
  assign wire9 = (((8'ha2) ?
                   wire3 : reg8[(4'h9):(4'h8)]) + $signed($signed(wire1)));
  assign wire10 = $signed($signed($unsigned((~|(wire2 ? wire0 : wire0)))));
  module11 modinst23 (.wire15(wire9), .wire16(wire3), .wire13(wire10), .wire12(wire1), .y(wire22), .wire14(wire0), .clk(clk));
  assign wire24 = $signed((wire1 ?
                    ((wire1 ? $unsigned(reg5) : ((8'hae) ? reg7 : wire9)) ?
                        ($unsigned(wire0) && $signed(wire22)) : $unsigned(reg4[(2'h3):(2'h2)])) : $unsigned(wire0)));
  assign wire25 = $unsigned($signed((~(|reg5))));
  assign wire26 = reg4[(3'h5):(1'h0)];
  assign wire27 = {(-wire0[(4'hd):(2'h2)]),
                    $signed($signed(($signed(reg4) != $unsigned((7'h41)))))};
endmodule

module module11  (y, clk, wire16, wire15, wire14, wire13, wire12);
  output wire  [(32'h40):(32'h0)] y;
  input wire  [(1'h0):(1'h0)] clk;
  input wire  [(2'h2):(1'h0)] wire16;
  input wire signed [(3'h4):(1'h0)] wire15;
  input wire signed [(5'h11):(1'h0)] wire14;
  input wire signed [(4'he):(1'h0)] wire13;
  input wire signed [(4'ha):(1'h0)] wire12;
  wire signed [(4'hf):(1'h0)] wire21;
  wire  [(4'hc):(1'h0)] wire20;
  wire  [(3'h7):(1'h0)] wire19;
  wire signed [(5'h11):(1'h0)] wire18;
  wire signed [(4'hc):(1'h0)] wire17;
  assign y = {wire21, wire20, wire19, wire18, wire17, (1'h0)};
  assign wire17 = $unsigned(wire14[(1'h1):(1'h0)]);
  assign wire18 = $unsigned(wire17);
  assign wire19 = ($signed(((^wire18[(4'hb):(2'h3)]) ^ ((8'hb9) ?
                        {(8'ha6), wire17} : $signed(wire16)))) ?
                    wire12[(2'h3):(1'h0)] : (+(+wire15[(2'h3):(2'h2)])));
  assign wire20 = (~|$signed(wire12));
  assign wire21 = (|$unsigned($signed(((-wire19) | wire15))));
endmodule
```

Example of generated Verilog by Verismith

- Bug of uninitialised reg in Yosys 0.8

Verilog generation

```verilog
// -*- mode: verilog -*-
module top #(parameter param30 = (8'hbb)) (y, clk, wire0, wire1, wire2, wire3);
  output [(32'hb7):(32'h0)] y;
  input [(1'h0):(1'h0)] clk;
  input signed [(5'h11):(1'h0)] wire0;
  input signed [(4'ha):(1'h0)] wire1;
  input [(4'hd):(1'h0)] wire2;
  input [(4'h8):(1'h0)] wire3;
  wire signed [(4'hb):(1'h0)] wire27;
  wire [(5'h15):(1'h0)] wire26;
  wire [(5'h10):(1'h0)] wire25;
  wire [(5'h13):(1'h0)] wire24;
  reg signed [(4'he):(1'h0)] reg4 = (1'h0);
  reg [(2'h3):(1'h0)] reg5 = (1'h0);
  reg [(5'h14):(1'h0)] reg6 = (1'h0);
  reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
  reg [(4'hd):(1'h0)] reg8 = (1'h0);
  wire [(4'hd):(1'h0)] wire9;
  wire [(4'he):(1'h0)] wire10;
  wire signed [(2'h2):(1'h0)] wire22;
  assign y = {wire27, wire26, wire25, wire24, reg4,
              reg5, reg6, reg7, reg8, wire9, wire10, wire22};
  always
    @(posedge clk) begin
      reg4 <=  wire1;
      if ($unsigned((~&(8'hb2))))
        begin
          reg5 <=  reg4;
          reg6 <=  wire1;
        end
      else
        begin
          reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
          reg6 <=  reg6;
        end
    end
  always @* begin
     reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 != (8'h9d))))) <<< ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                wire3 : $signed(wire1))) ? $unsigned(((^wire0) + $unsigned(wire3))) : (((reg5 * wire3) ?
                   wire1 : $unsigned(reg6)) ? {{reg4, wire2}} : (reg5[(1'h0):(1'h0)] ? $signed(reg4) : (~wire3)))));
     reg8 = (~^$unsigned(reg6));
  end
  assign wire9 = (((8'ha2) ?
                  wire3 : reg8[(4'h9):(4'h8)]) + $signed($signed(wire1)));
  assign wire10 = $signed($signed($unsigned((~|(wire2 ? wire0 : wire0)))));
  module11 modinst23 (.wire15(wire9), .wire16(wire3), .wire13(wire10), .wire12(wire1), .y(wire22), .wire14(wire0), .clk(clk));
  assign wire24 = $signed((wire1 ?
                   ((wire1 ? $unsigned(reg5) : ((8'hae) ? reg7 : wire9)) ?
                      ($unsigned(wire0) && $signed(wire22)) : $unsigned(reg4[(2'h3):(2'h2)])) : $unsigned(wire0)));
  assign wire25 = $unsigned($signed((~(|reg5))));
  assign wire26 = reg4[(3'h5):(1'h0)];
  assign wire27 = {(-wire0[(4'hd):(2'h2)]),
                   $signed($signed(($signed(reg4) != $unsigned((7'h41)))))};
endmodule

module module11  (y, clk, wire16, wire15, wire14, wire13, wire12);
  output wire  [(32'h40):(32'h0)] y;
  input wire  [(1'h0):(1'h0)] clk;
  input wire  [(2'h2):(1'h0)] wire16;
  input wire signed [(3'h4):(1'h0)] wire15;
  input wire signed [(5'h11):(1'h0)] wire14;
  input wire signed [(4'he):(1'h0)] wire13;
  input wire signed [(4'ha):(1'h0)] wire12;
  wire signed [(4'hf):(1'h0)] wire21;
  wire  [(4'hc):(1'h0)] wire20;
  wire  [(3'h7):(1'h0)] wire19;
  wire signed [(5'h11):(1'h0)] wire18;
  wire signed [(4'hc):(1'h0)] wire17;
  assign y = {wire21, wire20, wire19, wire18, wire17, (1'h0)};
  assign wire17 = $unsigned(wire14[(1'h1):(1'h0)]);
  assign wire18 = $unsigned(wire17);
  assign wire19 = ($signed(((^wire18[(4'hb):(2'h3)]) ^ ((8'hb9) ?
                      {(8'ha6), wire17} : $signed(wire16)))) ?
                   wire12[(2'h3):(1'h0)] : (+(+wire15[(2'h3):(2'h2)])));
  assign wire20 = (~|$signed(wire12));
  assign wire21 = (|$unsigned($signed(((-wire19) | wire15))));
endmodule
```

Example of generated Verilog by Verismith

- Bug of uninitialised reg in Yosys 0.8
- Random module items in the body

Verilog generation

```verilog
// -*- mode: verilog -*-
module top #(parameter param30 = (8'hbb)) (y, clk, wire0, wire1, wire2, wire3);
  output [(32'hb7):(32'h0)] y;
  input [(1'h0):(1'h0)] clk;
  input signed [(5'h11):(1'h0)] wire0;
  input signed [(4'ha):(1'h0)] wire1;
  input [(4'hd):(1'h0)] wire2;
  input [(4'h8):(1'h0)] wire3;
  wire signed [(4'hb):(1'h0)] wire27;
  wire [(5'h15):(1'h0)] wire26;
  wire [(5'h10):(1'h0)] wire25;
  wire [(5'h13):(1'h0)] wire24;
  reg signed [(4'he):(1'h0)] reg4 = (1'h0);
  reg [(2'h3):(1'h0)] reg5 = (1'h0);
  reg [(5'h14):(1'h0)] reg6 = (1'h0);
  reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
  reg [(4'hd):(1'h0)] reg8 = (1'h0);
  wire [(4'hd):(1'h0)] wire9;
  wire [(4'he):(1'h0)] wire10;
  wire signed [(2'h2):(1'h0)] wire22;
  assign y = {wire27, wire26, wire25, wire24, reg4,
              reg5, reg6, reg7, reg8, wire9, wire10, wire22};
  always
    @(posedge clk) begin
      reg4 <=  wire1;
      if ($unsigned((~&(8'hb2))))
        begin
          reg5 <=  reg4;
          reg6 <=  wire1;
        end
      else
        begin
          reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
          reg6 <=  reg6;
        end
    end
  always @* begin
      reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 != (8'h9d))))) <<< ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                 wire3 : $signed(wire1))) ? $unsigned(((^wire0) + $unsigned(wire3))) : (((reg5 * wire3) ?
                   wire1 : $unsigned(reg6)) ? {{reg4, wire2}} : (reg5[(1'h0):(1'h0)] ? $signed(reg4) : (~wire3)))));
      reg8 = (~^$unsigned(reg6));
  end
  assign wire9 = (((8'ha2) ?
                   wire3 : reg8[(4'h9):(4'h8)]) + $signed($signed(wire1)));
  assign wire10 = $signed($signed($unsigned((~|(wire2 ? wire0 : wire0)))));
  module11 modinst23 (.wire15(wire9), .wire16(wire3), .wire13(wire10), .wire12(wire1), .y(wire22), .wire14(wire0), .clk(clk));
  assign wire24 = $signed((wire1 ?
                   ((wire1 ? $unsigned(reg5) : ((8'hae) ? reg7 : wire9)) ?
                     ($unsigned(wire0) && $signed(wire22)) : $unsigned(reg4[(2'h3):(2'h2)])) : $unsigned(wire0)));
  assign wire25 = $unsigned($signed((~(|reg5))));
  assign wire26 = reg4[(3'h5):(1'h0)];
  assign wire27 = {(-wire0[(4'hd):(2'h2)]),
                   $signed($signed(($signed(reg4) != $unsigned((7'h41)))))};
endmodule

module module11  (y, clk, wire16, wire15, wire14, wire13, wire12);
  output wire  [(32'h40):(32'h0)] y;
  input wire  [(1'h0):(1'h0)] clk;
  input wire  [(2'h2):(1'h0)] wire16;
  input wire signed [(3'h4):(1'h0)] wire15;
  input wire signed [(5'h11):(1'h0)] wire14;
  input wire signed [(4'he):(1'h0)] wire13;
  input wire signed [(4'ha):(1'h0)] wire12;
  wire signed [(4'hf):(1'h0)] wire21;
  wire  [(4'hc):(1'h0)] wire20;
  wire  [(3'h7):(1'h0)] wire19;
  wire signed [(5'h11):(1'h0)] wire18;
  wire signed [(4'hc):(1'h0)] wire17;
  assign y = {wire21, wire20, wire19, wire18, wire17, (1'h0)};
  assign wire17 = $unsigned(wire14[(1'h1):(1'h0)]);
  assign wire18 = $unsigned(wire17);
  assign wire19 = ($signed(((^wire18[(4'hb):(2'h3)]) ^ ((8'hb9) ?
                     {(8'ha6), wire17} : $signed(wire16)))) ?
                   wire12[(2'h3):(1'h0)] : (+(+wire15[(2'h3):(2'h2)])));
  assign wire20 = (~|$signed(wire12));
  assign wire21 = (|$unsigned($signed(((-wire19) | wire15))));
endmodule
```

Example of generated Verilog by Verismith

- Bug of uninitialised reg in Yosys 0.8

- Random module items in the body
- Assignment of the internal state to the output

Verilog generation

```verilog
// -*- mode: verilog -*-
module top #(parameter param30 = (8'hbb)) (y, clk, wire0, wire1, wire2, wire3);
  output [(32'hb7):(32'h0)] y;
  input [(1'h0):(1'h0)] clk;
  input signed [(5'h11):(1'h0)] wire0;
  input signed [(4'ha):(1'h0)] wire1;
  input [(4'hd):(1'h0)] wire2;
  input [(4'h8):(1'h0)] wire3;
  wire signed [(4'hb):(1'h0)] wire27;
  wire [(5'h15):(1'h0)] wire26;
  wire [(5'h10):(1'h0)] wire25;
  wire [(5'h13):(1'h0)] wire24;
  reg signed [(4'he):(1'h0)] reg4 = (1'h0);
  reg [(2'h3):(1'h0)] reg5 = (1'h0);
  reg [(5'h14):(1'h0)] reg6 = (1'h0);
  reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
  reg [(4'hd):(1'h0)] reg8 = (1'h0);
  wire [(4'hd):(1'h0)] wire9;
  wire [(4'he):(1'h0)] wire10;
  wire signed [(2'h2):(1'h0)] wire22;
  assign y = {wire27, wire26, wire25, wire24, reg4,
              reg5, reg6, reg7, reg8, wire9, wire10, wire22};
  always
    @(posedge clk) begin
      reg4 <=  wire1;
      if ($unsigned((~&(8'hb2))))
        begin
          reg5 <=  reg4;
          reg6 <=  wire1;
        end
      else
        begin
          reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
          reg6 <=  reg6;
        end
    end
  always @* begin
     reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 != (8'h9d))))) <<< ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                wire3 : $signed(wire1))) ? $unsigned(((^wire0) + $unsigned(wire3))) : (((reg5 * wire3) ?
                  wire1 : $unsigned(reg6)) ? {{reg4, wire2}} : (reg5[(1'h0):(1'h0)] ? $signed(reg4) : (~wire3)))));
     reg8 = (~^$unsigned(reg6));
  end
  assign wire9 = (((8'ha2) ?
                  wire3 : reg8[(4'h9):(4'h8)]) + $signed($signed(wire1)));
  assign wire10 = $signed($signed($unsigned((~|(wire2 ? wire0 : wire0)))));
  module11 modinst23 (.wire15(wire9), .wire16(wire3), .wire13(wire10), .wire12(wire1), .y(wire22), .wire14(wire0), .clk(clk));
  assign wire24 = $signed((wire1 ?
                  ((wire1 ? $unsigned(reg5) : ((8'hae) ? reg7 : wire9)) ?
                    ($unsigned(wire0) && $signed(wire22)) : $unsigned(reg4[(2'h3):(2'h2)])) : $unsigned(wire0)));
  assign wire25 = $unsigned($signed((~(|reg5))));
  assign wire26 = reg4[(3'h5):(1'h0)];
  assign wire27 = {(-wire0[(4'hd):(2'h2)]),
                  $signed($signed(($signed(reg4) != $unsigned((7'h41)))))};
endmodule

module module11  (y, clk, wire16, wire15, wire14, wire13, wire12);
  output wire  [(32'h40):(32'h0)] y;
  input wire  [(1'h0):(1'h0)] clk;
  input wire  [(2'h2):(1'h0)] wire16;
  input wire signed [(3'h4):(1'h0)] wire15;
  input wire signed [(5'h11):(1'h0)] wire14;
  input wire signed [(4'he):(1'h0)] wire13;
  input wire signed [(4'ha):(1'h0)] wire12;
  wire signed [(4'hf):(1'h0)] wire21;
  wire  [(4'hc):(1'h0)] wire20;
  wire  [(3'h7):(1'h0)] wire19;
  wire signed [(5'h11):(1'h0)] wire18;
  wire signed [(4'hc):(1'h0)] wire17;
  assign y = {wire21, wire20, wire19, wire18, wire17, (1'h0)};
  assign wire17 = $unsigned(wire14[(1'h1):(1'h0)]);
  assign wire18 = $unsigned(wire17);
  assign wire19 = ($signed(((^wire18[(4'hb):(2'h3)]) ^ ((8'hb9) ?
                      {(8'ha6), wire17} : $signed(wire16)))) ?
                  wire12[(2'h3):(1'h0)] : (+(+wire15[(2'h3):(2'h2)])));
  assign wire20 = (~|$signed(wire12));
  assign wire21 = (|$unsigned($signed(((-wire19) | wire15))));
endmodule
```

Example of generated Verilog by Verismith

- Bug of uninitialised reg in Yosys 0.8
- Random module items in the body
- Assignment of the internal state to the output
- Definition of wires and initialisation of regs

Generation of the body

```
always
  @(posedge clk) begin
    reg4 <=  wire1;
    if ($unsigned((~&(8'hb2))))
      begin
        reg5 <=  reg4;
        reg6 <=  wire1;
      end
    else
      begin
        reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
        reg6 <=  reg6;
      end
  end
always @* begin
     reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 !=
     ↪  (8'h9d))))) <<< ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                wire3 : $signed(wire1))) ? $unsigned(((^wire0) +
                ↪  $unsigned(wire3))) : (((reg5 * wire3) ?
                    wire1 : $unsigned(reg6)) ? {{reg4, wire2}} :
                    ↪  (reg5[(1'h0):(1'h0)] ? $signed(reg4) :
                    ↪  (~wire3)))));
     reg8 = (~^$unsigned(reg6));
end
```

Generate Verilog node-by-node to:

- Ensure determinism
- Generate behavioural constructs
- Avoid logic loops

Generation of the body

```
always
  @(posedge clk) begin
    reg4 <=  wire1;
    if ($unsigned((~&(8'hb2))))
      begin
        reg5 <=  reg4;
        reg6 <=  wire1;
      end
    else
      begin
        reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
        reg6 <=  reg6;
      end
  end
always @* begin
     reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 !=
     ↪  (8'h9d))))) <<< ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
               wire3 : $signed(wire1))) ? $unsigned(((^wire0) +
               ↪  $unsigned(wire3))) : (((reg5 * wire3) ?
                   wire1 : $unsigned(reg6)) ? {{reg4, wire2}} :
                   ↪  (reg5[(1'h0):(1'h0)] ? $signed(reg4) :
                   ↪  (~wire3)))));
     reg8 = (~^$unsigned(reg6));
end
```

Unsupported constructs:

- function and task definitions
- alternate ranges (`+:`, `-:`)

Internal State Assignment

```
assign y = {wire27, wire26, wire25, wire24, reg4,
            reg5, reg6, reg7, reg8, wire9, wire10, wire22};
```

Need to assign all the internal state to the output `y`.

- As all the wires and regs are assigned a value, this concatenation can never be undefined.
- Any changes in the internal state are reflected in `y`.

Internal State Assignment

```
assign y = ^{wire27, wire26, wire25, wire24, reg4,
             reg5, reg6, reg7, reg8, wire9, wire10, wire22};
```

Need to assign all the internal state to the output y.

- As all the wires and regs are assigned a value, this concatenation can never be undefined.
- Any changes in the internal state are reflected in y.
- Try to xor into 1 bit, however synthesis and equivalence checking time suffer

Initialisation

```
output [(32'hb7):(32'h0)] y;
input [(1'h0):(1'h0)] clk;
input signed [(5'h11):(1'h0)] wire0;
input signed [(4'ha):(1'h0)] wire1;
input [(4'hd):(1'h0)] wire2;
input [(4'h8):(1'h0)] wire3;
wire signed [(4'hb):(1'h0)] wire27;
wire [(5'h15):(1'h0)] wire26;
wire [(5'h10):(1'h0)] wire25;
wire [(5'h13):(1'h0)] wire24;
reg signed [(4'he):(1'h0)] reg4 = (1'h0);
reg [(2'h3):(1'h0)] reg5 = (1'h0);
reg [(5'h14):(1'h0)] reg6 = (1'h0);
reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
reg [(4'hd):(1'h0)] reg8 = (1'h0);
wire [(4'hd):(1'h0)] wire9;
wire [(4'he):(1'h0)] wire10;
wire signed [(2'h2):(1'h0)] wire22;
```

Initialisation

```
output [(32'hb7):(32'h0)] y;
input [(1'h0):(1'h0)] clk;
input signed [(5'h11):(1'h0)] wire0;
input signed [(4'ha):(1'h0)] wire1;
input [(4'hd):(1'h0)] wire2;
input [(4'h8):(1'h0)] wire3;
wire signed [(4'hb):(1'h0)] wire27;
wire [(5'h15):(1'h0)] wire26;
wire [(5'h10):(1'h0)] wire25;
wire [(5'h13):(1'h0)] wire24;
reg signed [(4'he):(1'h0)] reg4 = (1'h0);
reg [(2'h3):(1'h0)] reg5 = (1'h0);
reg [(5'h14):(1'h0)] reg6 = (1'h0);
reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
reg [(4'hd):(1'h0)] reg8 = (1'h0);
wire [(4'hd):(1'h0)] wire9;
wire [(4'he):(1'h0)] wire10;
wire signed [(2'h2):(1'h0)] wire22;
```

- Define the inputs and outputs of the module with random sizes.

Initialisation

```
output [(32'hb7):(32'h0)] y;
input [(1'h0):(1'h0)] clk;
input signed [(5'h11):(1'h0)] wire0;
input signed [(4'ha):(1'h0)] wire1;
input [(4'hd):(1'h0)] wire2;
input [(4'h8):(1'h0)] wire3;
wire signed [(4'hb):(1'h0)] wire27;
wire [(5'h15):(1'h0)] wire26;
wire [(5'h10):(1'h0)] wire25;
wire [(5'h13):(1'h0)] wire24;
reg signed [(4'he):(1'h0)] reg4 = (1'h0);
reg [(2'h3):(1'h0)] reg5 = (1'h0);
reg [(5'h14):(1'h0)] reg6 = (1'h0);
reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
reg [(4'hd):(1'h0)] reg8 = (1'h0);
wire [(4'hd):(1'h0)] wire9;
wire [(4'he):(1'h0)] wire10;
wire signed [(2'h2):(1'h0)] wire22;
```

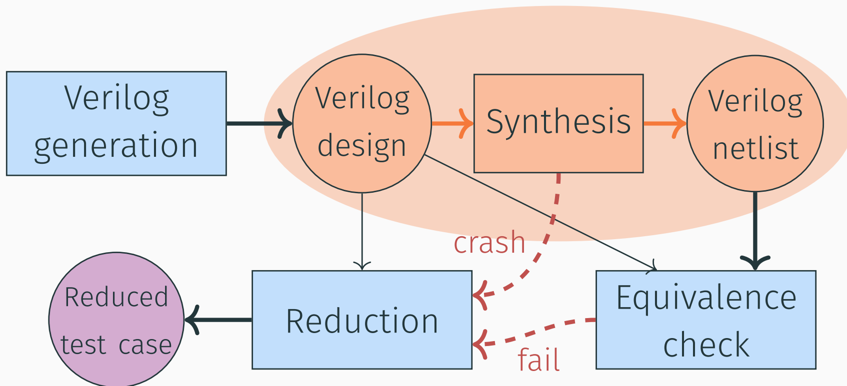
- Define the inputs and outputs of the module with random sizes.
- Define wires that get assigned in the module.

Initialisation

```
output [(32'hb7):(32'h0)] y;
input [(1'h0):(1'h0)] clk;
input signed [(5'h11):(1'h0)] wire0;
input signed [(4'ha):(1'h0)] wire1;
input [(4'hd):(1'h0)] wire2;
input [(4'h8):(1'h0)] wire3;
wire signed [(4'hb):(1'h0)] wire27;
wire [(5'h15):(1'h0)] wire26;
wire [(5'h10):(1'h0)] wire25;
wire [(5'h13):(1'h0)] wire24;
reg signed [(4'he):(1'h0)] reg4 = (1'h0);
reg [(2'h3):(1'h0)] reg5 = (1'h0);
reg [(5'h14):(1'h0)] reg6 = (1'h0);
reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
reg [(4'hd):(1'h0)] reg8 = (1'h0);
wire [(4'hd):(1'h0)] wire9;
wire [(4'he):(1'h0)] wire10;
wire signed [(2'h2):(1'h0)] wire22;
```

- Define the inputs and outputs of the module with random sizes.
- Define wires that get assigned in the module.
- Define and initialise regs to 0.

Run Outline: Synthesis

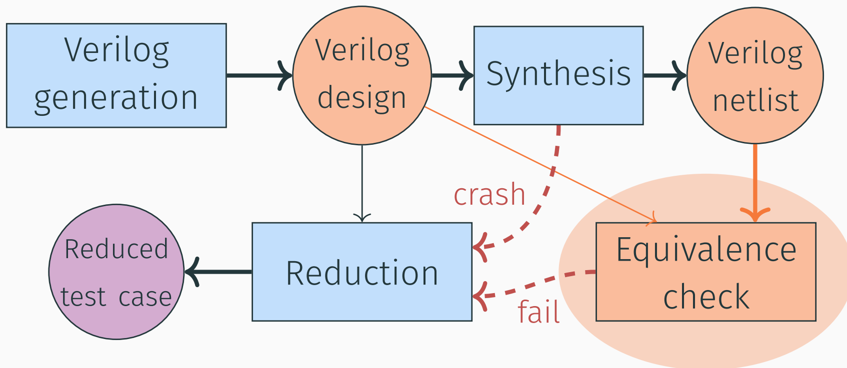


Yosys Synthesis

```
always
  @(posedge clk) begin
    reg4 <=  wire1;
    if ($unsigned((~&(8'hb2))))
      begin
        reg5 <=  reg4;
        reg6 <=  wire1;
      end
    else
      begin
        reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
        reg6 <=  reg6;
      end
  end
always @* begin
     reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 !=
      ↪  (8'h9d))))) <<< ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
               wire3 : $signed(wire1))) ? $unsigned(((^wire0) +
                ↪  $unsigned(wire3))) : (((reg5 * wire3) ?
                   wire1 : $unsigned(reg6)) ? {{reg4, wire2}} :
                    ↪  (reg5[(1'h0):(1'h0)] ? $signed(reg4) :
                    ↪  (~wire3)))));
     reg8 = (~^$unsigned(reg6));
end
```

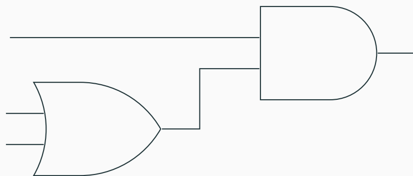
```
assign y[167] = ~_0116_;
assign y[168] = _0054_ ^ _0105_;
assign _0117_ = _0054_ & ~(wire0[4]);
assign y[169] = _0117_ ^ _0106_;
assign y[170] = _0055_ ^ _0107_;
assign _0118_ = _0055_ & ~(wire0[6]);
assign y[171] = _0118_ ^ _0108_;
assign _0119_ = _0055_ & ~(_0051_);
assign y[172] = _0119_ ^ _0109_;
assign _0120_ = _0119_ & ~(wire0[8]);
assign y[173] = _0120_ ^ _0110_;
assign y[174] = _0056_ ^ _0111_;
assign _0121_ = _0056_ & ~(wire0[10]);
assign y[175] = _0121_ ^ _0112_;
assign _0122_ = _0056_ & _0047_;
assign y[176] = _0122_ ^ _0113_;
assign _0123_ = ~(wire3[1] ^ wire1[1]);
assign _0124_ = wire3[0] & wire1[0];
assign wire9[1] = ~(_0124_ ^ _0123_);
assign _0125_ = ~(wire3[2] ^ wire1[2]);
assign _0126_ = _0124_ & ~(_0123_);
```

Run Outline: Equivalence Check



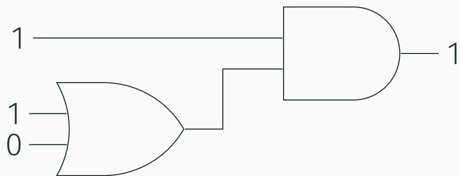
Equivalence check: What is an SMT solver?

- SAT solver with extra theories (e.g. Arrays to model memories)
- SAT solvers prove the satisfiability problem

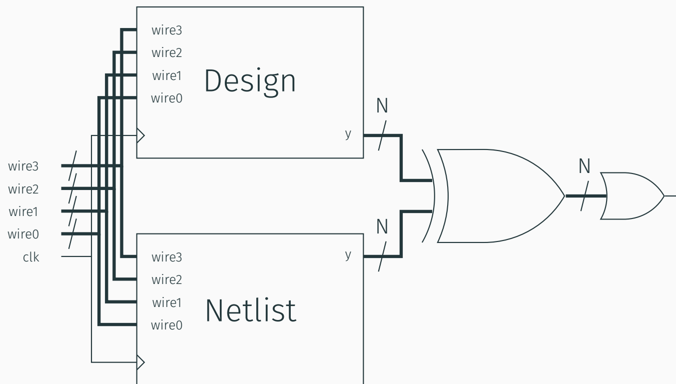


Equivalence check: What is an SMT solver?

- SAT solver with extra theories (e.g. Arrays to model memories)
- SAT solvers prove the satisfiability problem



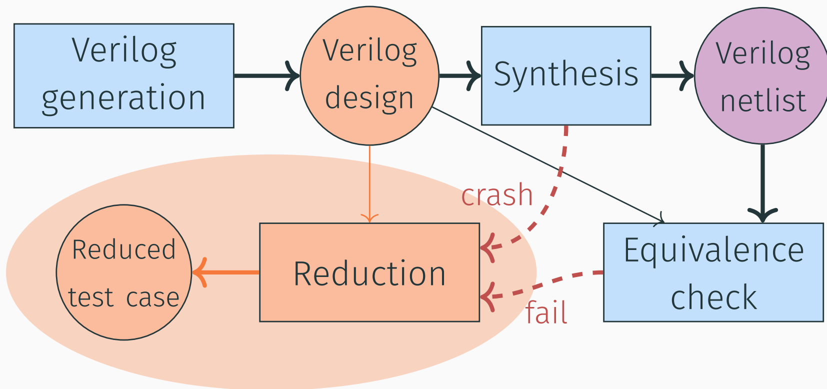
Equivalence check



Equivalence check done using an SMT solver (Z3) through Yosys

- Instantiate generated design with output `y_1`
- Instantiate synthesised netlist with output `y_2`
- Should be equal at every clock edge

Run Outline: Reduction



Reduction

```verilog
// -*- mode: verilog -*-
module top #(parameter param30 = (8'hbb)) (y, clk, wire0, wire1, wire2, wire3);
  output [(32'hb7):(32'h0)] y;
  input [(1'h0):(1'h0)] clk;
  input signed [(5'h11):(1'h0)] wire0;
  input signed [(4'ha):(1'h0)] wire1;
  input [(4'hd):(1'h0)] wire2;
  input [(4'h8):(1'h0)] wire3;
  wire signed [(4'hb):(1'h0)] wire27;
  wire [(5'h15):(1'h0)] wire26;
  wire [(5'h10):(1'h0)] wire25;
  wire [(5'h13):(1'h0)] wire24;
  reg signed [(4'he):(1'h0)] reg4 = (1'h0);
  reg [(2'h3):(1'h0)] reg5 = (1'h0);
  reg [(5'h14):(1'h0)] reg6 = (1'h0);
  reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
  reg [(4'hd):(1'h0)] reg8 = (1'h0);
  wire [(4'hd):(1'h0)] wire9;
  wire [(4'he):(1'h0)] wire10;
  wire signed [(2'h2):(1'h0)] wire22;
  assign y = {wire27, wire26, wire25, wire24, reg4,
              reg5, reg6, reg7, reg8, wire9, wire10, wire22};
  always
    @(posedge clk) begin
      reg4 <=  wire1;
      if ($unsigned((~&(8'hb2))))
        begin
          reg5 <=  reg4;
          reg6 <=  wire1;
        end
      else
        begin
          reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
          reg6 <=  reg6;
        end
    end
  always @* begin
      reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 != (8'h9d))))) <<<
       ↪  ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                wire3 : $signed(wire1))) ? $unsigned(((^wire0) + $unsigned(wire3))) :
                 ↪  (((reg5 * wire3) ?
                    wire1 : $unsigned(reg6)) ? {{reg4, wire2}} : (reg5[(1'h0):(1'h0)]
                     ↪  ? $signed(reg4) : (~wire3)))));
      reg8 = (~^$unsigned(reg6));
  end
  assign wire9 = (((8'ha2) ?
                   wire3 : reg8[(4'h9):(4'h8)]) + $signed($signed(wire1)));
  assign wire10 = $signed($signed($unsigned((~|(wire2 ? wire0 : wire0)))));
  module11 modinst23 (.wire15(wire9), .wire16(wire3), .wire13(wire10), .wire12(wire1),
   ↪  .y(wire22), .wire14(wire0), .clk(clk));
  assign wire24 = $signed((wire1 ?
                    ((wire1 ? $unsigned(reg5) : ((8'hae) ? reg7 : wire9)) ?
                      ($unsigned(wire0) && $signed(wire22)) :
                       ↪  $unsigned(reg4[(2'h3):(2'h2)])) : $unsigned(wire0)));
  assign wire25 = $unsigned($signed((~(|reg5))));
  assign wire26 = reg4[(3'h5):(1'h0)];
  assign wire27 = {(-wire0[(4'hd):(2'h2)]),
                    $signed($signed(($signed(reg4) != $unsigned((7'h41)))))};
endmodule

module module11  (y, clk, wire16, wire15, wire14, wire13, wire12);
  output wire  [(32'h40):(32'h0)] y;
  input wire  [(1'h0):(1'h0)] clk;
  input wire  [(2'h2):(1'h0)] wire16;
  input wire signed [(3'h4):(1'h0)] wire15;
  input wire signed [(5'h11):(1'h0)] wire14;
  input wire signed [(4'he):(1'h0)] wire13;
  input wire signed [(4'ha):(1'h0)] wire12;
  wire signed [(4'hf):(1'h0)] wire21;
  wire  [(4'hc):(1'h0)] wire20;
  wire  [(3'h7):(1'h0)] wire19;
  wire signed [(5'h11):(1'h0)] wire18;
  wire signed [(4'hc):(1'h0)] wire17;
  assign y = {wire21, wire20, wire19, wire18, wire17, (1'h0)};
  assign wire17 = $unsigned(wire14[(1'h1):(1'h0)]);
  assign wire18 = $unsigned(wire17);
  assign wire19 = ($signed(((^wire18[(4'hb):(2'h3)]) ^ ((8'hb9) ?
                      {(8'ha6), wire17} : $signed(wire16)))) ?
                    wire12[(2'h3):(1'h0)] : (+(+wire15[(2'h3):(2'h2)])));
  assign wire20 = (~|$signed(wire12));
  assign wire21 = (|$unsigned($signed(((-wire19) | wire15))));
endmodule
```

- Verilog has to be reduced to a minimal representation to identify the bug.
- Perform binary search on **syntax tree**.
- Traditional methods perform search on source code.

Reduction

```verilog
// -*- mode: verilog -*-
module top #(parameter param30 = (8'hbb)) (y, clk, wire0, wire1, wire2, wire3);
  output [(32'hb7):(32'h0)] y;
  input [(1'h0):(1'h0)] clk;
  input signed [(5'h11):(1'h0)] wire0;
  input signed [(4'ha):(1'h0)] wire1;
  input [(4'hd):(1'h0)] wire2;
  input [(4'h8):(1'h0)] wire3;
  wire signed [(4'hb):(1'h0)] wire27;
  wire [(5'h15):(1'h0)] wire26;
  wire [(5'h10):(1'h0)] wire25;
  wire [(5'h13):(1'h0)] wire24;
  reg signed [(4'he):(1'h0)] reg4 = (1'h0);
  reg [(2'h3):(1'h0)] reg5 = (1'h0);
  reg [(5'h14):(1'h0)] reg6 = (1'h0);
  reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
  reg [(4'hd):(1'h0)] reg8 = (1'h0);
  wire [(4'hd):(1'h0)] wire9;
  wire [(4'he):(1'h0)] wire10;
  wire signed [(2'h2):(1'h0)] wire22;
  assign y = {wire27, wire26, wire25, wire24, reg4,
              reg5, reg6, reg7, reg8, wire9, wire10, wire22};
  always
    @(posedge clk) begin
      reg4 <=  wire1;
      if ($unsigned((~&(8'hb2))))
        begin
          reg5 <=  reg4;
          reg6 <=  wire1;
        end
      else
        begin
          reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
          reg6 <=  reg6;
        end
    end
  always @* begin
      reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 != (8'h9d))))) <<<
      →  ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                wire3 : $signed(wire1))) ? $unsigned(((^wire0) + $unsigned(wire3))) :
                →  (((reg5 * wire3) ?
                    wire1 : $unsigned(reg6)) ? {{reg4, wire2}} : (reg5[(1'h0):(1'h0)]
                    →  ? $signed(reg4) : (~wire3)))));
      reg8 = (~^$unsigned(reg6));
  end
  assign wire9 = (((8'ha2) ?
                     wire3 : reg8[(4'h9):(4'h8)]) + $signed($signed(wire1)));
  assign wire10 = $signed($signed($unsigned((~|(wire2 ? wire0 : wire0)))));
  module11 modinst23 (.wire15(wire9), .wire16(wire3), .wire13(wire10), .wire12(wire1),
  →  .y(wire22), .wire14(wire0), .clk(clk));
  assign wire24 = $signed((wire1 ?
                    ((wire1 ? $unsigned(reg5) : ((8'hae) ? reg7 : wire9)) ?
                        ($unsigned(wire0) && $signed(wire22)) :
                        →  $unsigned(reg4[(2'h3):(2'h2)])) : $unsigned(wire0)));
  assign wire25 = $unsigned($signed((~(|reg5))));
  assign wire26 = reg4[(3'h5):(1'h0)];
  assign wire27 = {(-wire0[(4'hd):(2'h2)]),
                    $signed($signed(($signed(reg4) != $unsigned((7'h41)))))};
endmodule

module module11  (y, clk, wire16, wire15, wire14, wire13, wire12);
  output wire  [(32'h40):(32'h0)] y;
  input wire  [(1'h0):(1'h0)] clk;
  input wire  [(2'h2):(1'h0)] wire16;
  input wire signed [(3'h4):(1'h0)] wire15;
  input wire signed [(5'h11):(1'h0)] wire14;
  input wire signed [(4'he):(1'h0)] wire13;
  input wire signed [(4'ha):(1'h0)] wire12;
  wire signed [(4'hf):(1'h0)] wire21;
  wire  [(4'hc):(1'h0)] wire20;
  wire  [(3'h7):(1'h0)] wire19;
  wire signed [(5'h11):(1'h0)] wire18;
  wire signed [(4'hc):(1'h0)] wire17;
  assign y = {wire21, wire20, wire19, wire18, wire17, (1'h0)};
  assign wire17 = $unsigned(wire14[(1'h1):(1'h0)]);
  assign wire18 = $unsigned(wire17);
  assign wire19 = ($signed(((^wire18[(4'hb):(2'h3)]) ^ ((8'hb9) ?
                    {(8'ha6), wire17} : $signed(wire16)))) ?
                wire12[(2'h3):(1'h0)] : (+(+wire15[(2'h3):(2'h2)])));
  assign wire20 = (~|$signed(wire12));
  assign wire21 = (|$unsigned($signed(((-wire19) | wire15))));
endmodule
```

- Verilog has to be reduced to a minimal representation to identify the bug.
- Perform binary search on **syntax tree**.
- Traditional methods perform search on source code.

Reduction

```verilog
// -*- mode: verilog -*-
module top #(parameter param30 = (8'hbb)) (y, clk, wire0, wire1, wire2, wire3);
  output [(32'hb7):(32'h0)] y;
  input [(1'h0):(1'h0)] clk;
  input signed [(5'h11):(1'h0)] wire0;
  input signed [(4'ha):(1'h0)] wire1;
  input [(4'hd):(1'h0)] wire2;
  input [(4'h8):(1'h0)] wire3;
  wire signed [(4'hb):(1'h0)] wire27;
  wire [(5'h15):(1'h0)] wire26;
  wire [(5'h10):(1'h0)] wire25;
  wire [(5'h13):(1'h0)] wire24;
  reg signed [(4'he):(1'h0)] reg4 = (1'h0);
  reg [(2'h3):(1'h0)] reg5 = (1'h0);
  reg [(5'h14):(1'h0)] reg6 = (1'h0);
  reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
  reg [(4'hd):(1'h0)] reg8 = (1'h0);
  wire [(4'hd):(1'h0)] wire9;
  wire [(4'he):(1'h0)] wire10;
  assign y = {wire27, wire26, wire25, wire24, reg4,
              reg5, reg6, reg7, reg8, wire9, wire10};
  always
    @(posedge clk) begin
      reg4 <=  wire1;
      if ($unsigned((~&(8'hb2))))
        begin
          reg5 <=  reg4;
          reg6 <=  wire1;
        end
      else
        begin
          reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
          reg6 <=  reg6;
        end
    end
  always @* begin
       reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 != (8'h9d))))) <<<
        ↪  ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                  wire3 : $signed(wire1))) ? $unsigned(((^wire0) + $unsigned(wire3))) :
                   ↪  (((reg5 * wire3) ?
                      wire1 : $unsigned(reg6)) ? {{reg4, wire2}} : (reg5[(1'h0):(1'h0)]
                       ↪  ? $signed(reg4) : (~wire3)))));
       reg8 = (~^$unsigned(reg6));
  end
  assign wire9 = (((8'ha2) ?
                    wire3 : reg8[(4'h9):(4'h8)]) + $signed($signed(wire1)));
  assign wire10 = $signed($signed($unsigned((~|(wire2 ? wire0 : wire0)))));
  assign wire24 = $signed((wire1 ?
                      ((wire1 ? $unsigned(reg5) : ((8'hae) ? reg7 : wire9)) ?
                          ($unsigned(wire0) && 1'b0) : $unsigned(reg4[(2'h3):(2'h2)])) :
                           ↪  $unsigned(wire0)));
  assign wire25 = $unsigned($signed((~(|reg5))));
  assign wire26 = reg4[(3'h5):(1'h0)];
  assign wire27 = {(-wire0[(4'hd):(2'h2)]),
                      $signed($signed(($signed(reg4) != $unsigned((7'h41)))))};
endmodule
```

Search is performed on different levels of granularity:

- Modules
- Module items
- Statements inside always blocks
- Expressions

Reduction

```verilog
// -*- mode: verilog -*-
module top #(parameter param30 = (8'hbb)) (y, clk, wire0, wire1, wire2, wire3);
  output [(32'hb7):(32'h0)] y;
  input [(1'h0):(1'h0)] clk;
  input signed [(5'h11):(1'h0)] wire0;
  input signed [(4'ha):(1'h0)] wire1;
  input [(4'hd):(1'h0)] wire2;
  input [(4'h8):(1'h0)] wire3;
  wire signed [(4'hb):(1'h0)] wire27;
  wire [(5'h15):(1'h0)] wire26;
  wire [(5'h10):(1'h0)] wire25;
  wire [(5'h13):(1'h0)] wire24;
  reg signed [(4'he):(1'h0)] reg4 = (1'h0);
  reg [(2'h3):(1'h0)] reg5 = (1'h0);
  reg [(5'h14):(1'h0)] reg6 = (1'h0);
  reg signed [(5'h12):(1'h0)] reg7 = (1'h0);
  reg [(4'hd):(1'h0)] reg8 = (1'h0);
  wire [(4'hd):(1'h0)] wire9;
  wire [(4'he):(1'h0)] wire10;
  assign y = {wire27, wire26, wire25, wire24, reg4,
              reg5, reg6, reg7, reg8, wire9, wire10};
  always
    @(posedge clk) begin
      reg4 <=  wire1;
      if ($unsigned((~&(8'hb2))))
        begin
          reg5 <=  reg4;
          reg6 <=  wire1;
        end
      else
        begin
          reg5 <=  ($signed(reg7) ? wire2 : reg8[(4'h8):(2'h2)]);
          reg6 <=  reg6;
        end
    end
  always @* begin
       reg7 = ((~|((wire0 & {wire3, reg4}) | $unsigned((reg4 != (8'h9d))))) <<<
        ↪  ((wire1[(2'h2):(2'h2)] + ((~(8'ha7)) ?
                  wire3 : $signed(wire1))) ? $unsigned(((^wire0) + $unsigned(wire3))) :
                   ↪  (((reg5 * wire3) ?
                      wire1 : $unsigned(reg6)) ? {{reg4, wire2}} : (reg5[(1'h0):(1'h0)]
                       ↪  ? $signed(reg4) : (~wire3)))));
       reg8 = (~^$unsigned(reg6));
  end
  assign wire9 = (((8'ha2) ?
                     wire3 : reg8[(4'h9):(4'h8)]) + $signed($signed(wire1)));
  assign wire10 = $signed($signed($unsigned((~|(wire2 ? wire0 : wire0)))));
  assign wire24 = $signed((wire1 ?
                      ((wire1 ? $unsigned(reg5) : ((8'hae) ? reg7 : wire9)) ?
                          ($unsigned(wire0) && 1'b0) : $unsigned(reg4[(2'h3):(2'h2)])) :
                           ↪  $unsigned(wire0)));
  assign wire25 = $unsigned($signed((~(|reg5))));
  assign wire26 = reg4[(3'h5):(1'h0)];
  assign wire27 = {(-wire0[(4'hd):(2'h2)]),
                      $signed($signed(($signed(reg4) != $unsigned((7'h41)))))};
endmodule
```

Search is performed on different levels of granularity:

- Modules
- Module items
- Statements inside always blocks
- Expressions

Reduction

```
module top (y, clk, wire1);
   output wire [(32'hb7):(32'h0)] y;
   input wire [(1'h0):(1'h0)] clk;
   input wire signed [(4'ha):(1'h0)] wire1;
   reg signed [(4'he):(1'h0)] reg4 = (1'h0);
   assign y = {reg4};
   always
     @(posedge clk) reg4 <=  wire1;
endmodule
```

We then get a minimal testcase

Reduction

Input design

```
module top (y, clk, wire1);
   output wire [(32'hb7):(32'h0)] y;
   input wire [(1'h0):(1'h0)] clk;
   input wire signed [(4'ha):(1'h0)] wire1;
   reg signed [(4'he):(1'h0)] reg4 = (1'h0);
   assign y = {reg4};
   always
     @(posedge clk) reg4 <=  wire1;
endmodule
```

Yosys netlist

```
module top_1(y, clk, wire1);
   input clk;
   wire [1:0] reg4;
   input wire1;
   output [1:0] y;
   reg reg4_reg[0]  = 1'hx;
   always @(posedge clk)
     reg4_reg[0]  <= wire1;
   assign reg4[0] = reg4_reg[0] ;
   assign reg4[1] = reg4[0];
   assign y = { reg4[0], reg4[0] };
endmodule
```

Input design

```
module top (y, clk, wire1);
   output wire [(32'hb7):(32'h0)] y;
   input wire [(1'h0):(1'h0)] clk;
   input wire signed [(4'ha):(1'h0)] wire1;
   reg signed [(4'he):(1'h0)] reg4 = (1'h0);
   assign y = {reg4};
   always
     @(posedge clk) reg4 <=  wire1;
endmodule
```

Yosys netlist

```
module top_1(y, clk, wire1);
   input clk;
   wire [1:0] reg4;
   input wire1;
   output [1:0] y;
   reg reg4_reg[0]  = 1'b0;
   always @(posedge clk)
     reg4_reg[0]  <= wire1;
   assign reg4[0] = reg4_reg[0] ;
   assign reg4[1] = reg4[0];
   assign y = { reg4[0], reg4[0] };
endmodule
```

Experiments and Results

Bugs found

Tool	Total test cases	Failing test cases		Distinct failing test cases	Bug reports
Yosys 0.8	26400	7164	(27.1%)	≥ 1	0
Yosys 3333e00	51000	7224	(14.2%)	≥ 4	3
Yosys 70d0f38 (crash)	11	1	(9.09%)	1	1
Yosys 0.9	26400	611	(2.31%)	≥ 1	1
Vivado 2018.2	47992	1134	(2.36%)	≥ 5	3
Vivado 2018.2 (crash)	47992	566	(1.18%)	5	2
XST 14.7	47992	539	(1.12%)	≥ 2	0
Quartus Prime 19.2	80300	0	(0%)	0	0
Quartus Prime Lite 19.1	43	17	(39.5%)	1	0
Quartus Prime Lite 19.1 (No `$signed`)	137	0	(0%)	0	0
Icarus Verilog 10.3	26400	616	(2.33%)	≥ 1	1

- Summary of all the tests run over 18000 CPU hours

Bugs found

Tool	Total test cases	Failing test cases		Distinct failing test cases	Bug reports
Yosys 0.8	26400	7164	(27.1%)	≥ 1	0
Yosys 3333e00	51000	7224	(14.2%)	≥ 4	3
Yosys 70d0f38 (crash)	11	1	(9.09%)	1	1
Yosys 0.9	26400	611	(2.31%)	≥ 1	1
Vivado 2018.2	47992	1134	(2.36%)	≥ 5	3
Vivado 2018.2 (crash)	47992	566	(1.18%)	5	2
XST 14.7	47992	539	(1.12%)	≥ 2	0
Quartus Prime 19.2	80300	0	(0%)	0	0
Quartus Prime Lite 19.1	43	17	(39.5%)	1	0
Quartus Prime Lite 19.1 (No `$signed`)	137	0	(0%)	0	0
Icarus Verilog 10.3	26400	616	(2.33%)	≥ 1	1

- Quartus Prime Light failing because of the handling of `$signed`
- No crashes or failures found in Quartus Prime

Bugs found

Tool	Total test cases	Failing test cases		Distinct failing test cases	Bug reports
Yosys 0.8	26400	7164	(27.1%)	≥ 1	0
Yosys 3333e00	51000	7224	(14.2%)	≥ 4	3
Yosys 70d0f38 (crash)	11	1	(9.09%)	1	1
Yosys 0.9	26400	611	(2.31%)	≥ 1	1
Vivado 2018.2	47992	1134	(2.36%)	≥ 5	3
Vivado 2018.2 (crash)	47992	566	(1.18%)	5	2
XST 14.7	47992	539	(1.12%)	≥ 2	0
Quartus Prime 19.2	80300	0	(0%)	0	0
Quartus Prime Lite 19.1	43	17	(39.5%)	1	0
Quartus Prime Lite 19.1 (No `$signed`)	137	0	(0%)	0	0
Icarus Verilog 10.3	26400	616	(2.33%)	≥ 1	1

- Vivado was the only stable tool that crashed

Bugs found

Tool	Total test cases	Failing test cases		Distinct failing test cases	Bug reports
Yosys 0.8	26400	7164	(27.1%)	≥ 1	0
Yosys 3333e00	51000	7224	(14.2%)	≥ 4	3
Yosys 70d0f38 (crash)	11	1	(9.09%)	1	1
Yosys 0.9	26400	611	(2.31%)	≥ 1	1
Vivado 2018.2	47992	1134	(2.36%)	≥ 5	3
Vivado 2018.2 (crash)	47992	566	(1.18%)	5	2
XST 14.7	47992	539	(1.12%)	≥ 2	0
Quartus Prime 19.2	80300	0	(0%)	0	0
Quartus Prime Lite 19.1	43	17	(39.5%)	1	0
Quartus Prime Lite 19.1 (No `$signed`)	137	0	(0%)	0	0
Icarus Verilog 10.3	26400	616	(2.33%)	≥ 1	1

- Yosys improved quite a lot between versions
- Yosys 0.9 contains all the bug fixes that were submitted

Bugs found

Tool	Total test cases	Failing test cases		Distinct failing test cases	Bug reports
Yosys 0.8	26400	7164	(27.1%)	≥ 1	0
Yosys 3333e00	51000	7224	(14.2%)	≥ 4	3
Yosys 70d0f38 (crash)	11	1	(9.09%)	1	1
Yosys 0.9	26400	611	(2.31%)	≥ 1	1
Vivado 2018.2	47992	1134	(2.36%)	≥ 5	3
Vivado 2018.2 (crash)	47992	566	(1.18%)	5	2
XST 14.7	47992	539	(1.12%)	≥ 2	0
Quartus Prime 19.2	80300	0	(0%)	0	0
Quartus Prime Lite 19.1	43	17	(39.5%)	1	0
Quartus Prime Lite 19.1 (No `$signed`)	137	0	(0%)	0	0
Icarus Verilog 10.3	26400	616	(2.33%)	≥ 1	1

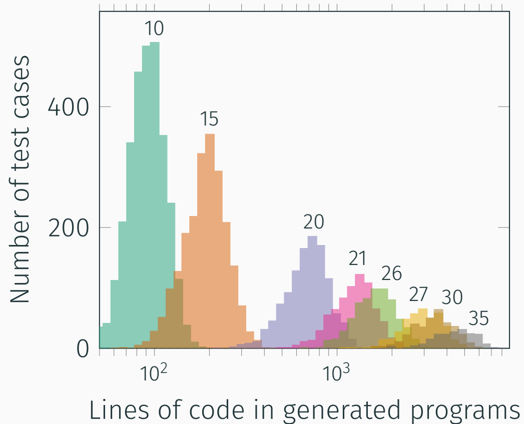
- Yosys development versions also tested to aid development

Bugs found

Tool	Total test cases	Failing test cases		Distinct failing test cases	Bug reports
Yosys 0.8	26400	7164	(27.1%)	≥ 1	0
Yosys 3333e00	51000	7224	(14.2%)	≥ 4	3
Yosys 70d0f38 (crash)	11	1	(9.09%)	1	1
Yosys 0.9	26400	611	(2.31%)	≥ 1	1
Vivado 2018.2	47992	1134	(2.36%)	≥ 5	3
Vivado 2018.2 (crash)	47992	566	(1.18%)	5	2
XST 14.7	47992	539	(1.12%)	≥ 2	0
Quartus Prime 19.2	80300	0	(0%)	0	0
Quartus Prime Lite 19.1	43	17	(39.5%)	1	0
Quartus Prime Lite 19.1 (No `$signed`)	137	0	(0%)	0	0
Icarus Verilog 10.3	26400	616	(2.33%)	≥ 1	1

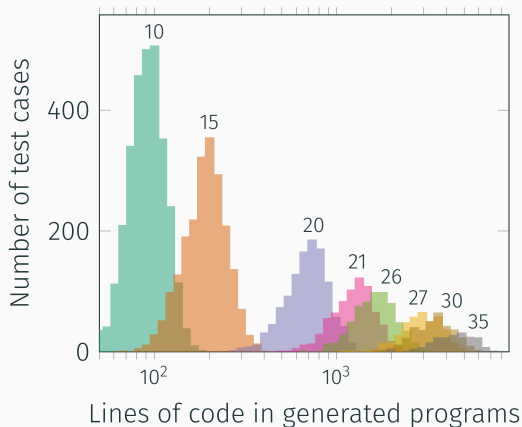
- Truncation bug in Icarus Verilog found while checking SMT counter examples

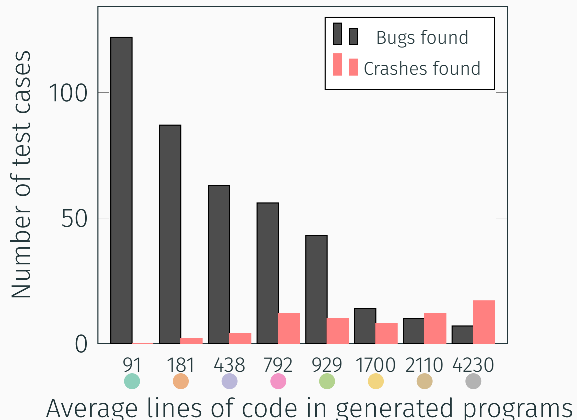
Efficiency at different Verilog sizes



- Each experiment was run over 3 days with Yosys, Vivado and XST

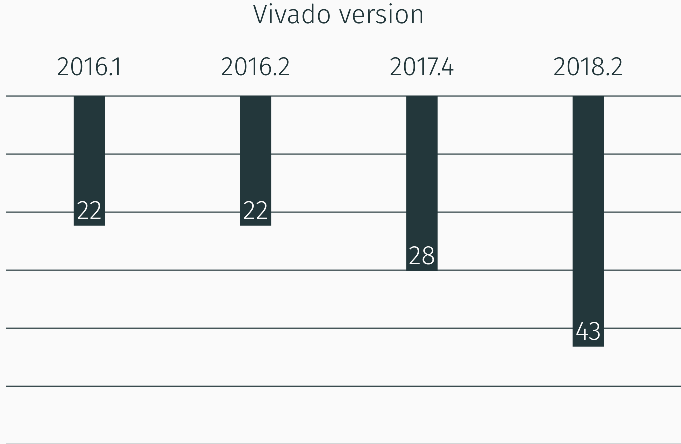
Efficiency at different Verilog sizes





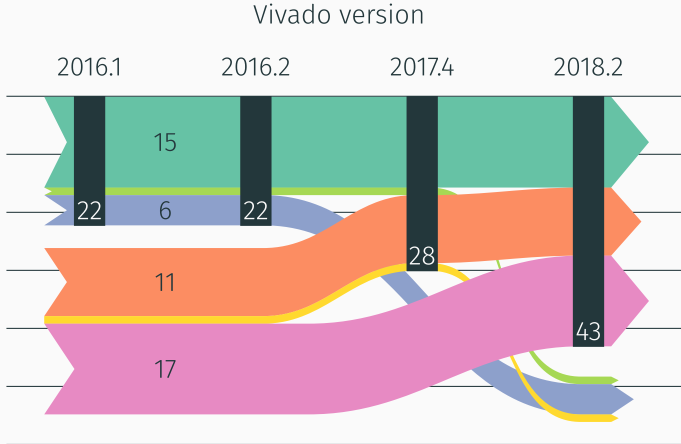
- Each experiment was run over 3 days with Yosys, Vivado and XST

Bugs found in Vivado over different versions



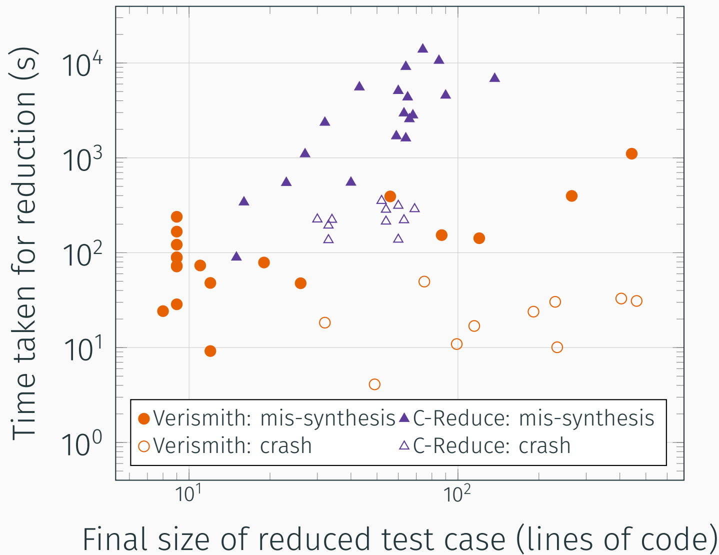
- Total number of failing testcases increase with versions
- This does not mean there are more bugs, just that they were more commonly found

Bugs found in Vivado over different versions



- Total number of failing testcases increase with versions
- This does not mean there are more bugs, just that they were more commonly found

Reduction efficiency



Difficulties we encountered

- Understanding the Verilog standards

Difficulties we encountered

- Understanding the Verilog standards
- Implementing missing modules in the netlist for device specific components

Difficulties we encountered

- Understanding the Verilog standards
- Implementing missing modules in the netlist for device specific components
  - Especially had problems with `dffeas` module in Quartus

Difficulties we encountered

- Understanding the Verilog standards
- Implementing missing modules in the netlist for device specific components
  - Especially had problems with `dffeas` module in Quartus
  - Also had problems with encrypted modules in Quartus which had to be disabled (e.g. multiply accumulate optimisations)

Difficulties we encountered

- Understanding the Verilog standards
- Implementing missing modules in the netlist for device specific components
  - Especially had problems with `dffeas` module in Quartus
  - Also had problems with encrypted modules in Quartus which had to be disabled (e.g. multiply accumulate optimisations)
- Time taken to perform synthesis and equivalence checking time
  - Difficult to fuzz tools that take a long time to finish

Summary

- Found and reported hard-to-find bugs so that these could be fixed before affecting users
- In general synthesis tools don't seem to be reliable enough as bugs were found in all of them except for Quartus Prime

11 unique bugs were found, reported and fixed by tool vendors.

Future work:

- Support a larger subset of Verilog
- Add controlled nondeterminism

Finding and Understanding Bugs in FPGA Synthesis Tools

Yann Herklotz, John Wickerson

Verismith Github[2]

Link to paper[3]

[2] https://github.com/ymherklotz/verismith
[3] https://yannherklotz.com/papers/fubfst_fpga2020.pdf

Motivating Bug 2: Vivado

```
module top (output [1:0] y,
            input clk,
            input [1:0] w0 );
   reg [1:0] r0 = 2'b0;
   reg [2:0] r1 = 3'b0;
   assign y = r1;
   always @(posedge clk) begin
      r0 <= 1'b1;
      if (r0)
        r1 <= r0 ?  w0[0:0]  : 1'b0;
      else r1 <= 3'b1;
   end
endmodule
```

Bug found in Vivado 2019.1.[4]

[4] https://forums.xilinx.com/t5/Synthesis/Vivado-2019-1-Bit-selection-synthesis-mismatch/td-p/982419

Motivating Bug 2: Vivado

```
module top (output [1:0] y,
            input clk,
            input [1:0] w0);
  reg [1:0] r0 = 2'b0;
  reg [2:0] r1 = 3'b0;
  assign y = r1;
  always @(posedge clk) begin
    r0 <= 1'b1;
    if (r0)
      r1 <= r0 ? w0[0:0] : 1'b0;
    else r1 <= 3'b1;
  end
endmodule
```

Bug found in Vivado 2019.1.[4]

- Assume `w0 = 2'b10`,

[4] https://forums.xilinx.com/t5/Synthesis/Vivado-2019-1-Bit-selection-synthesis-mismatch/td-p/982419

Motivating Bug 2: Vivado

```
module top (output [1:0] y,
            input clk,
            input [1:0] w0 );
  reg [1:0] r0 = 2'b0;
  reg [2:0] r1 = 3'b0;
  assign y = r1;
  always @(posedge clk) begin
    r0 <= 1'b1;
    if (r0)
      r1 <= r0 ? w0[0:0] : 1'b0;
    else r1 <= 3'b1;
  end
endmodule
```

Bug found in Vivado 2019.1.[4]

- Assume `w0 = 2'b10`,
- initialise `r0 = 2'b0`, `r1 = 3'b0`,

[4] https://forums.xilinx.com/t5/Synthesis/Vivado-2019-1-Bit-selection-synthesis-mismatch/td-p/982419

Motivating Bug 2: Vivado

```
module top (output [1:0] y,
            input clk,
            input [1:0] w0 );
  reg [1:0] r0 = 2'b0;
  reg [2:0] r1 = 3'b0;
  assign y = r1;
  always @(posedge clk) begin
    r0 <= 1'b1;
    if (r0)
      r1 <= r0 ? w0[0:0] : 1'b0;
    else r1 <= 3'b1;
  end
endmodule
```

Bug found in Vivado 2019.1.[4]

- Assume `w0 = 2'b10`,
- initialise `r0 = 2'b0`, `r1 = 3'b0`,
- first `clk` edge sets `r0 = 1'b1`, `r1 = 3'b1`,

[4] https://forums.xilinx.com/t5/Synthesis/Vivado-2019-1-Bit-selection-synthesis-mismatch/td-p/982419

Motivating Bug 2: Vivado

```
module top (output [1:0] y,
            input clk,
            input [1:0] w0 );
   reg [1:0] r0 = 2'b0;
   reg [2:0] r1 = 3'b0;
   assign y = r1;
   always @(posedge clk) begin
      r0 <= 1'b1;
      if (r0)
        r1 <= r0 ? w0[0:0] : 1'b0;
      else r1 <= 3'b1;
   end
endmodule
```

Bug found in Vivado 2019.1.[4]

- Assume `w0 = 2'b10`,
- initialise `r0 = 2'b0`, `r1 = 3'b0`,
- first `clk` edge sets `r0 = 1'b1`, `r1 = 3'b1`,
- next `clk` edge enters the `if` statement,

[4] https://forums.xilinx.com/t5/Synthesis/Vivado-2019-1-Bit-selection-synthesis-mismatch/td-p/982419

Motivating Bug 2: Vivado

```
module top (output [1:0] y,
            input clk,
            input [1:0] w0 );
   reg [1:0] r0 = 2'b0;
   reg [2:0] r1 = 3'b0;
   assign y = r1;
   always @(posedge clk) begin
      r0 <= 1'b1;
      if (r0)
         r1 <= r0 ? w0[0:0] : 1'b0;
      else r1 <= 3'b1;
   end
endmodule
```

Bug found in Vivado 2019.1.[4]

- Assume `w0 = 2'b10`,
- initialise `r0 = 2'b0`, `r1 = 3'b0`,
- first `clk` edge sets `r0 = 1'b1`, `r1 = 3'b1`,
- next `clk` edge enters the `if` statement,
- sets `r1 = w0[0:0] = 3'b0`
  Vivado returns `r1 = w0[0:0] = 3'b010`

[4] https://forums.xilinx.com/t5/Synthesis/Vivado-2019-1-Bit-selection-synthesis-mismatch/td-p/982419