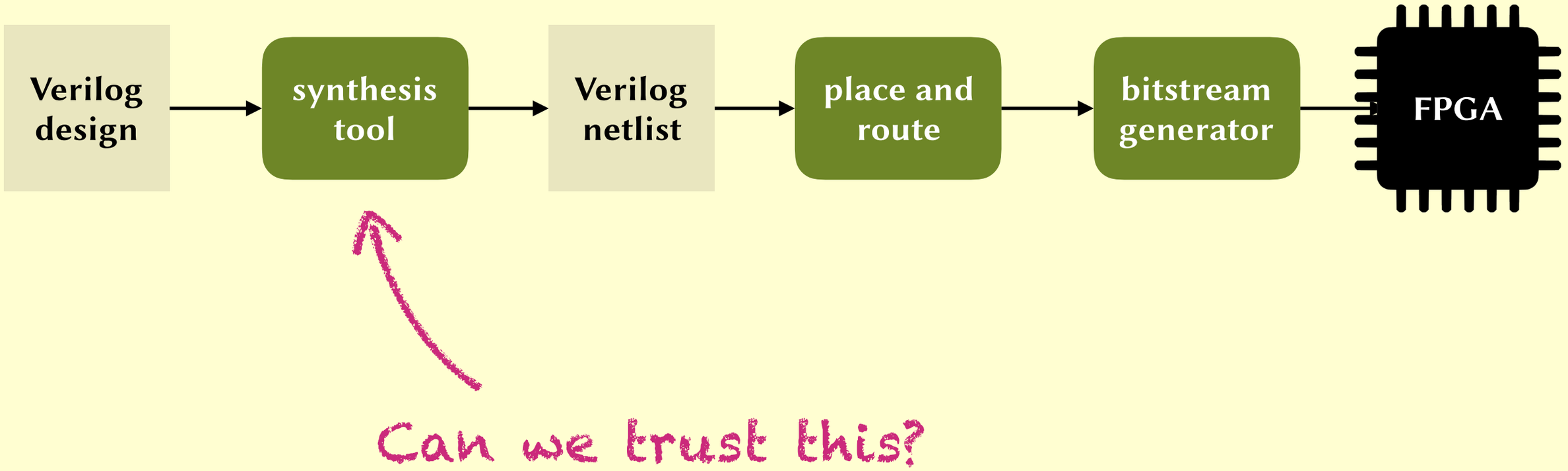
Finding and understanding bugs in FPGA synthesis tools

Yann Herklotz and **John Wickerson**

Department of Electrical and Electronic Engineering
Imperial College London

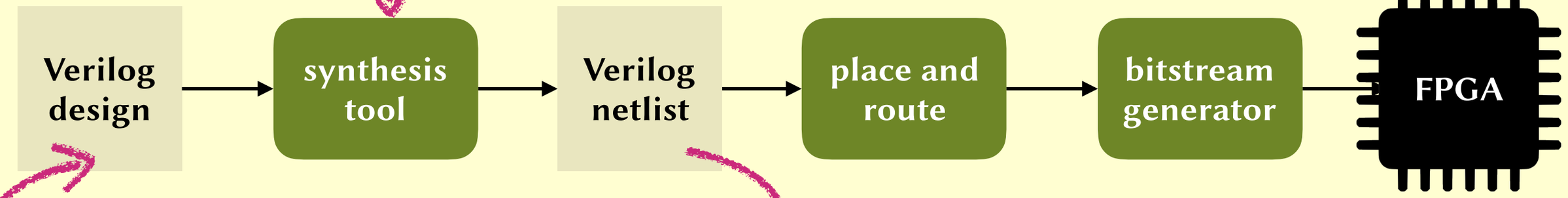
27 November 2019

The FPGA synthesis toolflow



Xilinx Vivado
Yosys
Intel Quartus
Xilinx XST

Verismith




```
module top #(parameter param0 = 5'h9e23848124)
(y, clk, wire0, wire1, wire2, wire3);
  // *** Declarations ***
  output wire [(1'h0):(1'h0)] y;
  input wire [(1'h0):(1'h0)] clk;
  input wire [(3'h6):(1'h0)] wire0;
  input wire [(4'ha):(1'h0)] wire1;
  input wire signed [(4'ha):(1'h0)] wire2;
  input wire [(4'hb):(1'h0)] wire3;
  reg [(3'h2):(1'h0)] reg20 = (1'h0);
  reg [(3'h5):(1'h0)] reg19 = (1'h0);
  reg [(3'h4):(1'h0)] reg18 = (1'h0);
  reg [(2'h2):(1'h0)] reg17 = (1'h0);
  reg [(4'ha):(1'h0)] reg16 = (1'h0);
  reg signed [(4'h9):(1'h0)] reg15 = (1'h0);
  wire [(3'h6):(1'h0)] wire5;
  wire [(2'h3):(1'h0)] wire4;
  // *** Assign output ***
  assign y =
    {reg20,reg19,reg18,reg17,reg16,reg15,wire5,wire4};
  // *** Random module items ***
  assign wire4 = (((~wire5) ? ((((15'h9ecc51592fdeb04)
    ? reg17[(5'h1f):(2'h2)] : (reg18 ? wire2 : wire0))
    ? $unsigned(((-2'ha73a956341f45c0) << reg18)) :
    wire1[(4'hb):(3'h7)]) - reg18) :
    reg15[(4'h9):(3'h7)]) >>> $unsigned($signed((
    reg16[(4'he):(3'h7)] ? ((wire5 && reg16) &&
    {reg15, reg15, wire3}) : (reg18 ? (~&wire3) :
    (-39'ha7a1419cd4ea34a))))));
  assign wire5 = $signed(((wire2 ? (
    (-8'h5e411249da4f335) ? (4'hb2fa97daeae9ff) :
    wire1) : (wire4 ? wire2 : wire1)) ?
    $signed(wire3) : (((7'hbac46141008d14)} >>>
    (&wire0))));
  always @(posedge clk) begin
    for (reg15 = (1'h0); (reg15 < (2'h2)); reg15 =
        (reg15 + (1'h1))) begin
      if (((wire3 == (~(reg16 + wire1))) >=
          {$signed(wire0[(2'h2):(1'h0)])}))
        reg16 <=  ($unsigned($unsigned(wire1)) <
          wire3[(1'h1):(1'h1)]);
      else reg16 <= $unsigned(reg17[(2'h2):(2'h0)]);
      reg17 <=  wire3[(1'h0):(1'h0)];
    end
    reg18 <= $signed(({wire0} ~^ wire3));
  end
  always @(posedge clk) begin
    if (wire3[(4'h9):(3'h6)])
      reg19 = $signed($unsigned(wire1)) <<
        $unsigned({wire1});
    reg20 <=  (((((~|wire3), $unsigned(reg19)} ?
      reg16 : reg15[(2'h2):(1'h1)]),
      (~&((wire0 ? wire3 : reg17) ~^ reg18))}
      || ((~&(wire3[(4'hb):(4'h9)] ? wire4 : (+wire5)))));
  end
endmodule
```

```
module top #(parameter param0 = 5'h9e23848124)
(y, clk, wire0, wire1, wire2, wire3);
  // *** Declarations ***
  output wire [(1'h0):(1'h0)] y;
  input wire [(1'h0):(1'h0)] clk;
  input wire [(3'h6):(1'h0)] wire0;
  input wire [(4'ha):(1'h0)] wire1;
  input wire signed [(4'ha):(1'h0)] wire2;
  input wire [(4'hb):(1'h0)] wire3;
  reg [(3'h2):(1'h0)] reg20 = (1'h0);
  reg [(3'h5):(1'h0)] reg19 = (1'h0);
  reg [(3'h4):(1'h0)] reg18 = (1'h0);
  reg [(2'h2):(1'h0)] reg17 = (1'h0);
  reg [(4'ha):(1'h0)] reg16 = (1'h0);
  reg signed [(4'h9):(1'h0)] reg15 = (1'h0);
  wire [(3'h6):(1'h0)] wire5;
  wire [(2'h3):(1'h0)] wire4;
  // *** Assign output ***
  assign y =
    {reg20,reg19,reg18,reg17,reg16,reg15,wire5,wire4};
  // *** Random module items ***
  assign wire4 = (((~wire5) ? ((((15'h9ecc51592fdeb04)
    ? reg17[(5'h1f):(2'h2)] : (reg18 ? wire2 : wire0))
    ? $unsigned(((-2'ha73a956341f45c0) << reg18)) :
    wire1[(4'hb):(3'h7)]) - reg18) :
    reg15[(4'h9):(3'h7)]) >>> $unsigned($signed((
    reg16[(4'he):(3'h7)] ? ((wire5 && reg16) &&
    {reg15, reg15, wire3}) : (reg18 ? (~&wire3) :
    (-39'ha7a1419cd4ea34a))))));
  assign wire5 = $signed(((wire2 ? (
    (-8'h5e411249da4f335) ? (4'hb2fa97daeae9ff) :
    wire1) : (wire4 ? wire2 : wire1)) ?
    $signed(wire3) : (((7'hbac46141008d14)} >>>
    (&wire0))));
  always @(posedge clk) begin
    for (reg15 = (1'h0); (reg15 < (2'h2)); reg15 =
        (reg15 + (1'h1))) begin
      if (((wire3 == (~(reg16 + wire1))) >=
          {$signed(wire0[(2'h2):(1'h0)])}))
        reg16 <=  ($unsigned($unsigned(wire1)) <
          wire3[(1'h1):(1'h1)]);
      else reg16 <= $unsigned(reg17[(2'h2):(2'h0)]);
      reg17 <=  wire3[(1'h0):(1'h0)];
    end
    reg18 <= $signed(({wire0} ~^ wire3));
  end
  always @(posedge clk) begin
    if (wire3[(4'h9):(3'h6)])
      reg19 = $signed($unsigned(wire1)) <<
        $unsigned({wire1});
    reg20 <=  (((((~|wire3), $unsigned(reg19)} ?
      reg16 : reg15[(2'h2):(1'h1)]),
      (~&((wire0 ? wire3 : reg17) ~^ reg18))}
      || ((~&(wire3[(4'hb):(4'h9)] ? wire4 : (+wire5)))));
  end
endmodule
```

equivalence checker

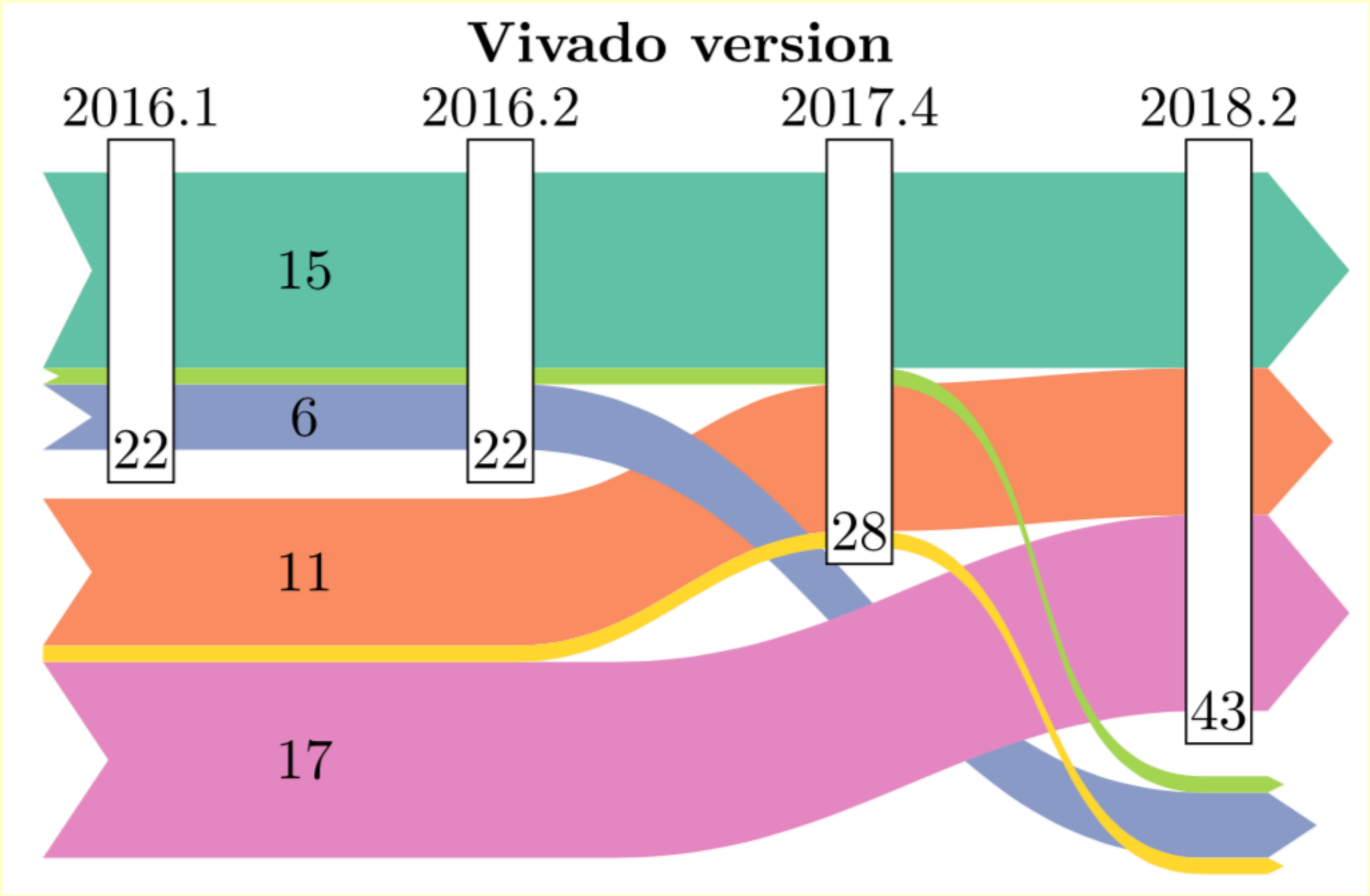
testcase reducer

Repeat x50000

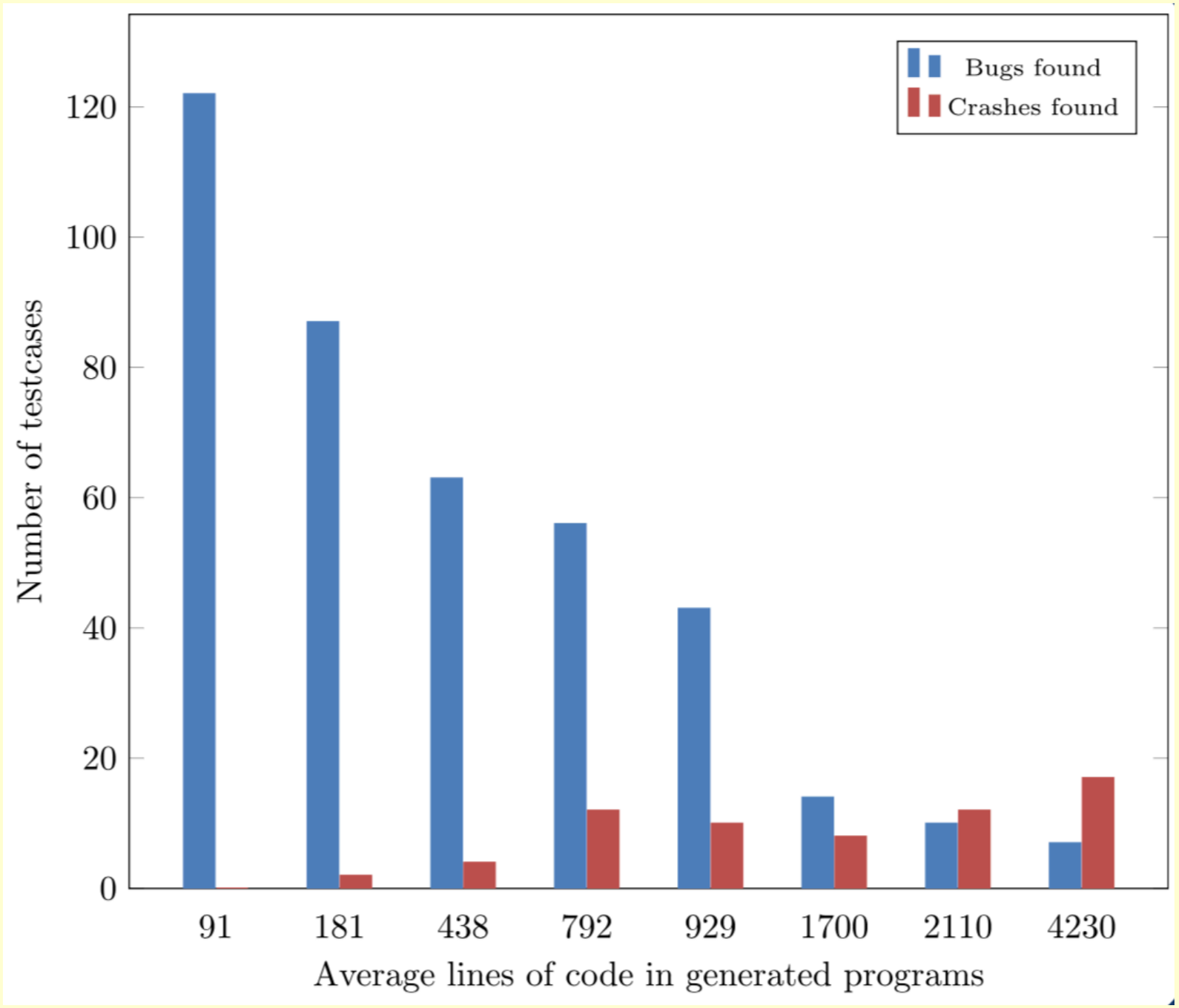
Results

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 18.2	47992	1134 (2.36%)	≥ 5	3
Vivado 18.2 (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

Tracking bugs over time



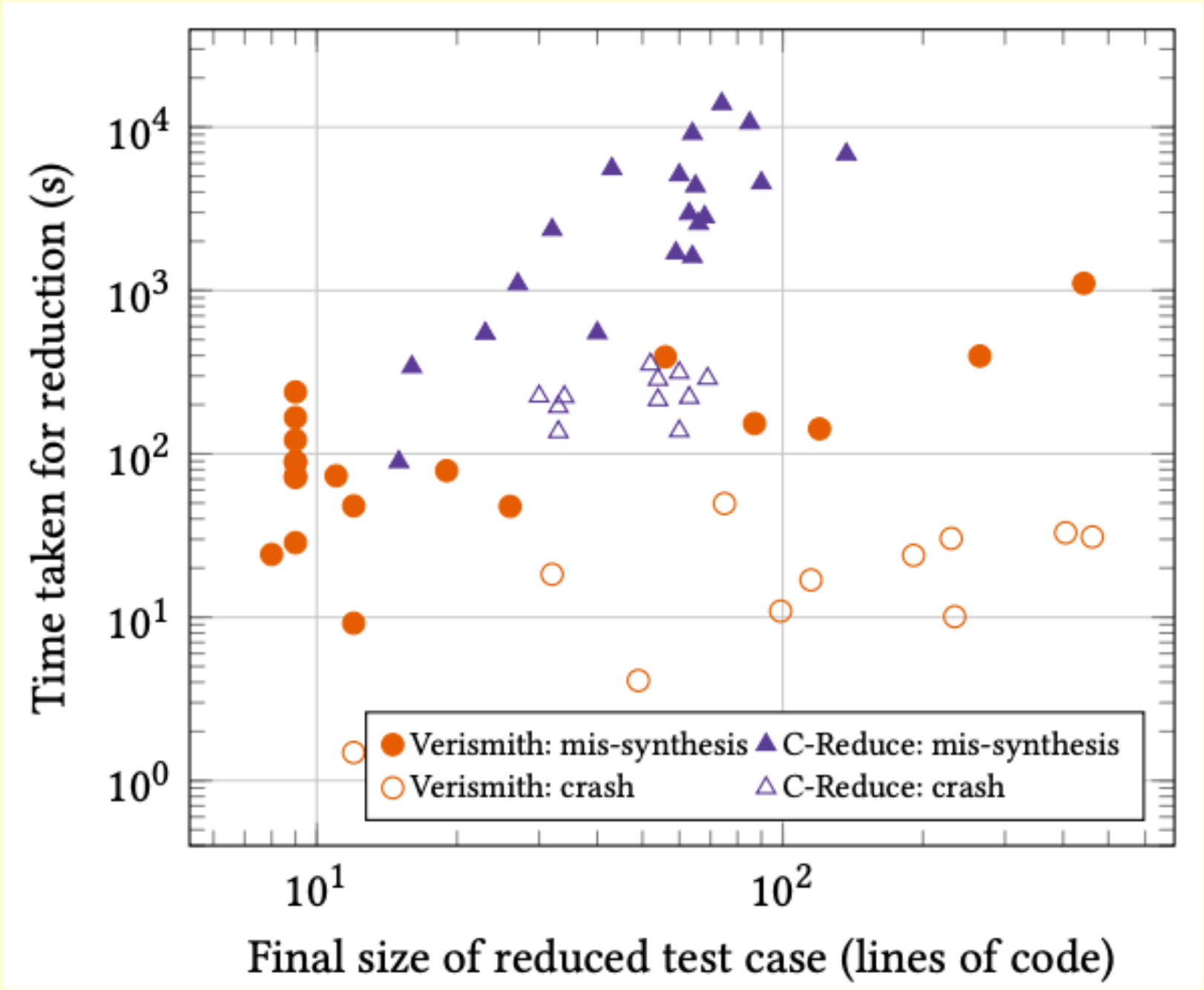
Generating good testcases




```
[probability]
  expr.binary = 5
  expr.concatenation = 3
  expr.number = 1
  expr.rangeselect = 5
  expr.signed = 5
  expr.string = 0
  expr.ternary = 5
  expr.unary = 5
  expr.unsigned = 5
  expr.variable = 5
  moditem.assign = 5
  moditem.combinational = 1
  moditem.instantiation = 1
  moditem.sequential = 1
  statement.blocking = 0
  statement.conditional = 1
  statement.forloop = 0
  statement.nonblocking = 3

[property]
  module.depth = 2
  module.max = 5
  output.combine = false
  sample.method = "random"
  sample.size = 10
  size = 20
  statement.depth = 3
```

Evaluating reduction



Do these bugs matter?

XILINX

Re: Vivado 2019.1 Bit selection synthesis mismatch

Does this bug affect any priori Vivado versions. i. 2018.*?

This looks to me to be a rather critical bug. @ymherklotz - did you try your test on any other Vivado version?

Next steps

- Test ASIC synthesis tools
- Metamorphic testing
- Test other stages of the hardware synthesis process, e.g. place-and-route