Finding and Understanding Bugs in FPGA Synthesis Tools

Yann Herklotz* and John Wickerson†
Imperial College London
*yann.herklotz15@imperial.ac.uk, †j.wickerson@imperial.ac.uk

FPGA '20 paper

Motivation

Problem:

- FPGAs are becoming more common in cloud computing for application-specific hardware accelerators.
- Synthesis tools are unreliable, which could lead to the FPGA behaving incorrectly, even if the design was proven to behave correctly.

Solution:

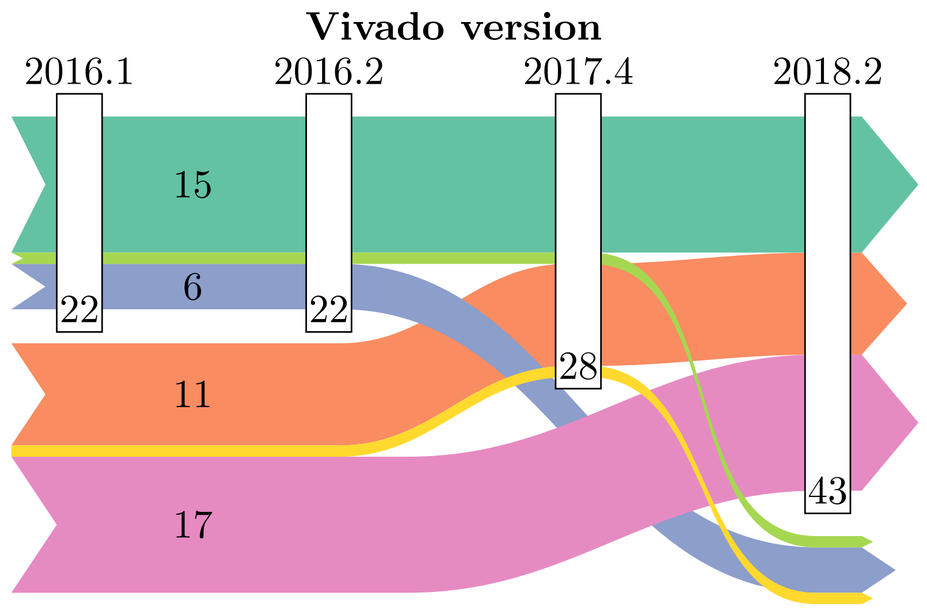
- We can test the tools using the following method:
  1. Random Verilog generation.
  2. Synthesise to a netlist.
  3. Prove equivalence of netlist and design.
  4. Reduce failed testcases.
- Any bugs found can then be reported to the tool vendors and will hopefully be fixed before end users are affected.
- To help in reporting the bugs, failing testcases are also automatically reduced to a minimal representation.

Results

This was implemented in a tool called **Verismith**[1] and was used to test the main FPGA synthesis tools:

- **5** bugs found in **Vivado**.
- **5** bugs found and fixed in **Yosys**.
- **2** bugs found in **XST**.
- **1** bug found in **Quartus**.

Stability in Vivado:



Tracking the same set of testcases across four versions of Vivado. The white rectangles indicate the total number of failing testcases per version. Each ribbon tracks a particular group of testcases. The interleaving of ribbons shows how bugs may have been introduced or fixed in each version.

[1] https://github.com/ymherklotz/verismith

1. Verilog generation

The first step is to generate the random Verilog. The main goals of the Verilog generation are the following:

- Generate correct and deterministic Verilog.
- Generate efficient Verilog in terms of synthesis time and time taken to perform an equivalence check.
- Generate complex behavioural Verilog.

Verilog design

2. Synthesis

Synthesis is the process being tested. Currently the following tools are supported:

- **Vivado**
- **Quartus**
- **Yosys**
- **XST**

The design is passed to the tools and should output a Verilog netlist that is functionally equivalent to the input.

Definitions for any modules that are instantiated in the netlist are manually implemented so that it simulates properly.

Verilog netlist

3. Equivalence check

We perform an equivalence check between the original design and the synthesised netlist. This is done by proving that the output of the design is always the same as the output of the netlist.

The steps to perform the equivalence check are the following:

- Combine the Verilog design and netlist into a testbench with an assertion saying that the outputs should always be equivalent.
- Convert testbench to SMT-LIBv2.
- Check the equivalence by passing it to an SMT solver.

Crash

Fail

4. Reduction

If the synthesis crashes or the equivalence check fails, the original design is reduced to locate the cause.

As synthesis and the equivalence checks are extremely slow, minimising the number of reductions necessary is the main concern.

```
module top (y, clk, w1);
  output y;
  input clk;
  input signed [1:0] w1;
  reg r1 = 1'b0;
  assign y = r1;
  always @(posedge clk)
    if ({-1'b1 == w1})
      r1 <= 1'b1;
endmodule
```

Mis-synthesis bug found in Vivado 2019.1. Mistakenly outputs 1'b1 when input is 2'b01.

Imperial College London