Validating the Redundancy Assumption for HDL from Code Clone's Perspective

Jianjun Xu[1], Jiayu He[1], Jingyan Zhang[1], Deheng Yang[1],

Jiang Wu[1], Xiaoguang Mao[1]

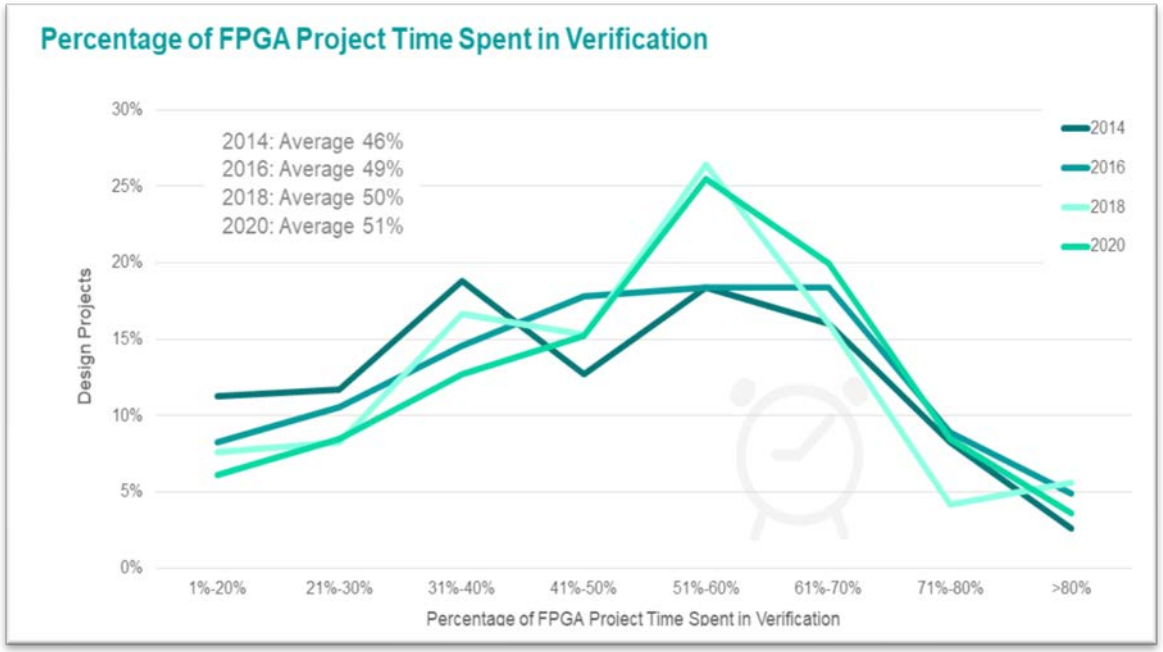
[1] National University of Defense Technology

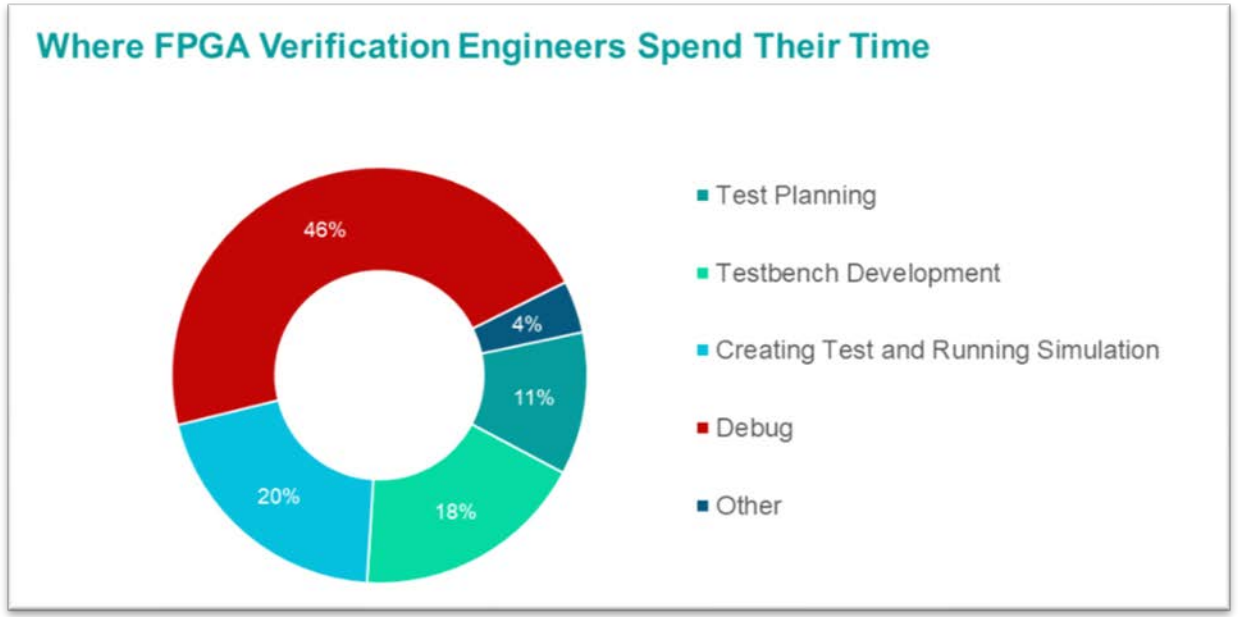
Outline

1. Motivation
2. Approach
3. Research Questions
4. Results
5. Conclusion

Motivation

[Debugging Cost] Over a half of FPGA project time is spent on verification in 2020. Almost a half of verification time is spent on debugging[1]

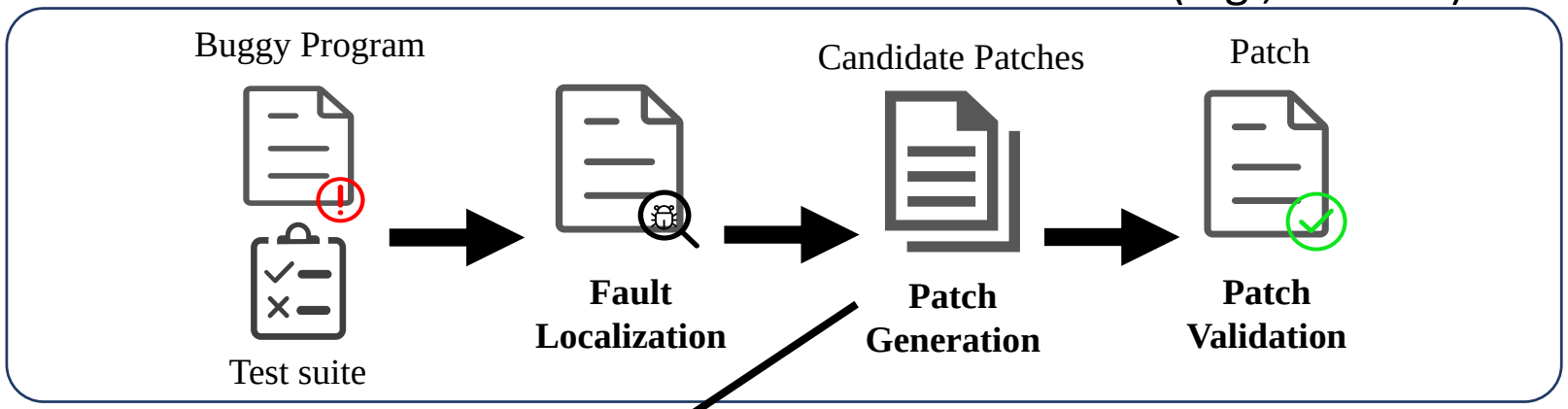




[1] *2020 Wilson Research Group Functional Verification Study*

Motivation

Automated program repair (APR) techniques aim to repair defects with no human intervention, which has also attracted the attention of researchers in hardware (e.g., CirFix [2]).



CirFix

- Genetic Programming
- Pre-defined repair templates

- Most APR techniques use existing code fragments to generate candidate patches.
- The redundancy assumption has not been validated in hardware description languages.

[2] *Ahmad, Hammad, Yu Huang, and Westley Weimer. "Cirfix: automatically repairing defects in hardware design code." Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022.*

Motivation

Redundancy Assumption: code fragments for bug fixing already exist elsewhere in the code.

```
Commit 2f686fe62486a5abedd81331d083c86016350803
--- a/rtl/zipsystem.v
+++ b/rtl/zipsystem.v
@@ -328,30 +334,32 @@ module zipsystem(i_clk,
     i_rst,
   reg cmd_reset, cmd_halt, cmd_step,
       cmd_clear_pf_cache;
   reg [5:0] cmd_addr;
   wire [3:0] cpu_dbg_cc;
- assign dbg_cmd_write = (dbg_cyc)&&(dbg_stb)&&(
     dbg_we)&&(~dbg_addr);
+ assign dbg_cmd_write = (dbg_stb)&&(dbg_we)&&(!
     dbg_addr);
   ...
```

Listing 1: Example of bug fixing in zipcpu.

Donor Code

- The concrete code fragments that are used to synthesize a patch (e.g., replace or insert).

Redundancy Assumption

Redundancy assumption is that donor code can be found in original program. Formally,

$$\exists f \in DN, f \in F$$

f refers to code fragments, DN refers to the donor codes, F refers to the buggy program

Motivation

Redundancy Assumption: code fragments for bug fixing already exist elsewhere in the code.

```
Commit 2f686fe62486a5abedd81331d083c86016350803
--- a/rtl/zipsystem.v
+++ b/rtl/zipsystem.v
@@ -328,30 +334,32 @@ module zipsystem(i_clk,
    i_rst,
  reg cmd_reset, cmd_halt, cmd_step,
      cmd_clear_pf_cache;
  reg [5:0] cmd_addr;
  wire [3:0] cpu_dbg_cc;
- assign dbg_cmd_write = (dbg_cyc)&&(dbg_stb)&&(
    dbg_we)&&(~dbg_addr);
+ assign dbg_cmd_write = (dbg_stb)&&(dbg_we)&&(!
    dbg_addr);
  ...
```

Listing 1: Example of bug fixing in zipcpu.

Donor Code

- The concrete code fragments that are used to synthesize a patch (e.g., replace or insert).

Redundancy Assumption

- Can donor code be found from the original buggy program?
- How to search for donor code?

Approach

Therefore, we propose **VClone** to extract and detect the clones of real-world bug fixes in Verilog automatically

- To validate the ground truth of donor code being in the original program;
- To explore the donor code availability from code clone's perspective;

Code Clone

- Type 3 clones refer to copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.

Redundancy Assumption

Donor code can be found from the code clones of buggy code. Formally,

$$\exists f \in DN, f \in F \wedge f \in CodeClone(f_b)$$

f refers to code fragments, DN refers to the donor codes, F refers to the buggy program, f_b refers to the buggy code snippet

We then perform code clone detection of bug fixes in Verilog projects

- the code context of buggy statements and their clones are similar
- the clones contain the donor code

Approach

```
326  //
327  wire  cpu_break, dbg_cmd_write;
328  reg  cmd_reset, cmd_halt, cmd_step,
         cmd_clear_pf_cache;
329  reg [5:0] cmd_addr;
330  wire [3:0] cpu_dbg_cc;
331  assign dbg_cmd_write =
         (dbg_cyc)&&(dbg_stb)&&(dbg_we)&&( dbg_addr);
332  //
333  initial cmd_reset = 1'b1;
334  always @(posedge i_clk)
335   cmd_reset <= ((dbg_cmd_write)&&(dbg_idata[6])
         );
336  //
```

Listing 2: Buggy code snippet in `zipsystem.v`.

```
142  // register.
143  //
144  wire  cpu_break, dbg_cmd_write;
145  reg  cmd_reset, cmd_halt, cmd_step,
         cmd_clear_pf_cache;
146  reg [4:0] cmd_addr;
147  wire [3:0] cpu_dbg_cc;
148  assign dbg_cmd_write =
         (dbg_stb)&&(dbg_we)&&(!dbg_addr);
149  //
150  // Always start us off with an initial reset
151  //
152  initial cmd_reset = 1'b1;
```

Listing 3: One of candidate code clones in `zipbones.v`.

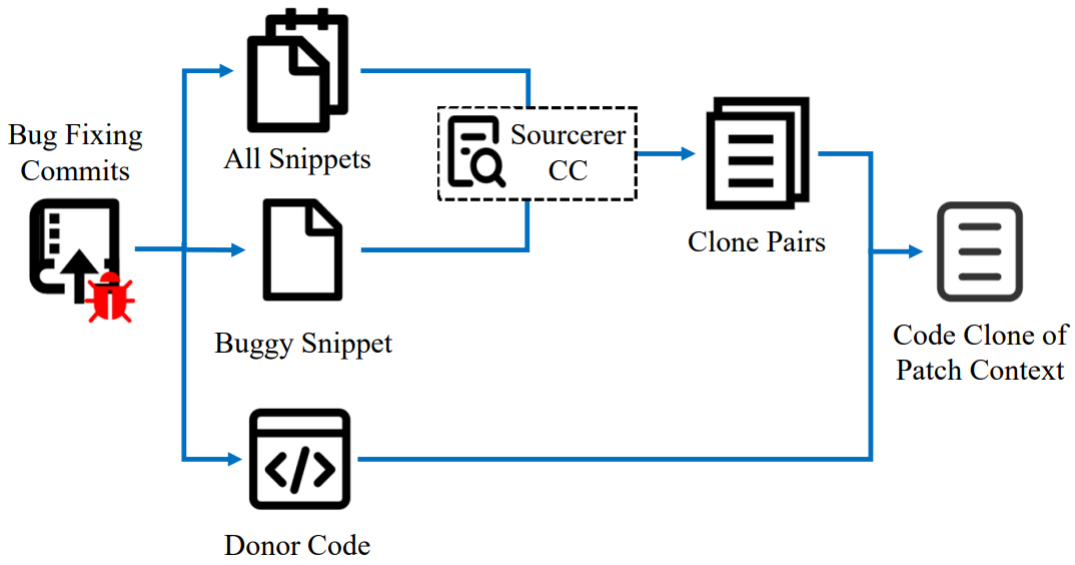
Buggy Code Snippet

- The code snippet in the buggy program that involves code changes.

Code Clone of Patch Context

- Code clone of *buggy code snippet* that contains the donor code.

Approach



1. Retrieving changed code.
2. Fragmenting buggy program.
3. Detecting code clones.
4. Analyzing code clones.

Table 1: Information of subject programs in our study.

Projects	Source Files	LOC	Commits	
			All	**Identified**
picorv32	14	5,404	592	**55**
e200_opensource	147	77,857	196	**5**
wujian100_open	43	107,305	41	**0**
hw	407	941,371	102	**0**
verilog-ethernet	301	109,689	1,026	**85**
hdl	534	140,681	6,203	**618**
amiga2000-gfxcard	25	10,468	153	**19**
corundum	769	359,329	2,719	**193**
zipcpu	39	24,290	569	**74**
oh	389	40,289	1,584	**304**
serv	60	5,013	361	**28**
miaow	184	33,978	93	**24**
Total	2,912	1,855,674	13,639	**1,405**

LOC: # lines of code. **All:** # of all available commits in a project. **Identified:** # of identified bug fixing commits.

Research Questions

1. **[Redundancy Assumption]**
   What extent are code clones of patch context in Verilog?

2. **[Distribution]**
   How is the distribution about code clones of patch context?

RQ1: redundancy assumption

Table 2: Code Clone of Patch Context in 9 Open-Source Projects.

Projects	ALL Snippets	Global		Local	
		Available Snippets	Percentage	Available Snippets	Percentage
picorv32	10	1	10.00%	0	0.00%
e200_opensource	6	0	0.00%	0	0.00%
verilog-ethernet	953	197	20.67%	15	1.57%
hdl	2,569	532	20.71%	446	17.36%
amiga2000-gfxcard	160	10	6.25%	6	3.75%
zipcpu	133	26	19.55%	14	10.53%
oh	1,163	138	11.87%	117	10.06%
serv	109	4	3.67%	4	3.67%
miaow	30	1	3.33%	0	0.00%
Total	5,133	909	17.71%	602	11.77%

ALL Snippets refer to the code snippets introducing new code fragments. **Available Snippets** refer to the cloned code snippets that contain code clone of patch context.

- **Global:** a buggy block is globally redundant if the corresponding code clone of patch context can be found elsewhere in **the program**;
- **Local:** a buggy block is locally redundant if the corresponding code clone of patch context is from **the same file**.

RQ1: redundancy assumption

Global Scope

Projects	ALL Snippets	Global	
		Available Snippets	Percentage
picorv32	10	1	10.00%
e200_opensource	6	0	0.00%
verilog-ethernet	953	197	20.67%
hdl	2,569	532	20.71%
amiga2000-gfxcard	160	10	6.25%
zipcpu	133	26	19.55%
oh	1,163	138	11.87%
serv	109	4	3.67%
miaow	30	1	3.33%
Total	5,133	909	17.71%

Finding

- On average, the clone coverage of bug fixes at global scope is up to 17.71%. The result not only validates the redundancy assumption in HDLs, but provides a direction for APR techniques toward HDLs to search for donor code.

RQ1: redundancy assumption

Local Scope

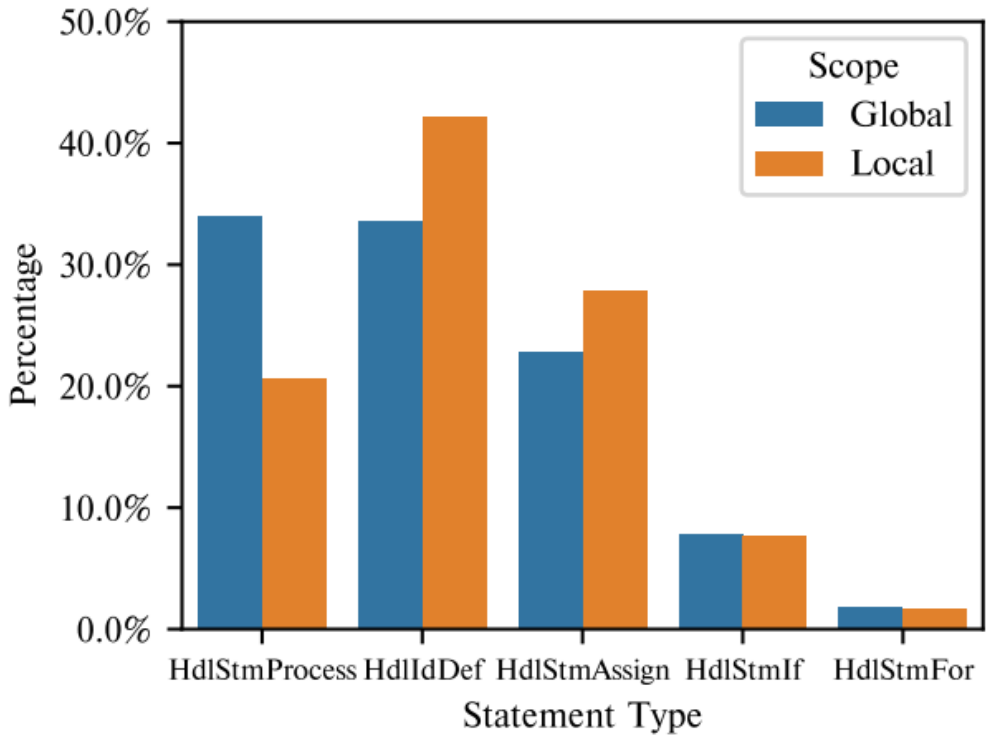
Projects	ALL Snippets	Local	
		Available Snippets	Percentage
picorv32	10	0	0.00%
e200_opensource	6	0	0.00%
verilog-ethernet	953	15	1.57%
hdl	2,569	446	17.36%
amiga2000-gfxcard	160	6	3.75%
zipcpu	133	14	10.53%
oh	1,163	117	10.06%
serv	109	4	3.67%
miaow	30	0	0.00%
Total	5,133	602	11.77%

Finding

- There is still a significant percentage (11.77%) of code clones of bug fix at local scope overall, but the local scope can reduce the search space effectively.

RQ2: Distribution

Distribution of Clones in Statement Types



Finding

- The redundant bug fixes are mainly distributed among a few statement types (i.e., *HdlStmProcess, HdlIdDef, HdlStmAssign, HdlStmIf and HdlStmFor*), which can motivate APR techniques targeting a specific type of statements

Conclusion

1. An empirical validation of the redundancy assumption on open-source Verilog projects.
2. An analysis of distribution of redundant bug fixes.
3. An automatic tool to extract and detect the clones of real-world bug fixes in Verilog

VIRTUAL ISPD 2023

32nd International Symposium on

Thank you for your attention!

Physical Design

March 26-29, 2023