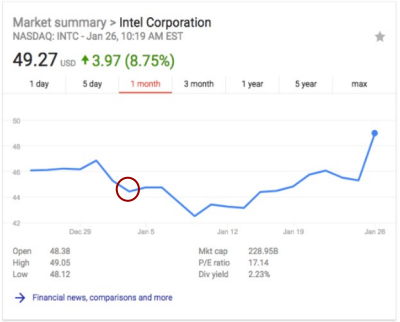
CirFix: Automatically Repairing Defects in Hardware Design Code

Hammad Ahmad, Yu Huang, Westley Weimer
University of Michigan, Ann Arbor, MI

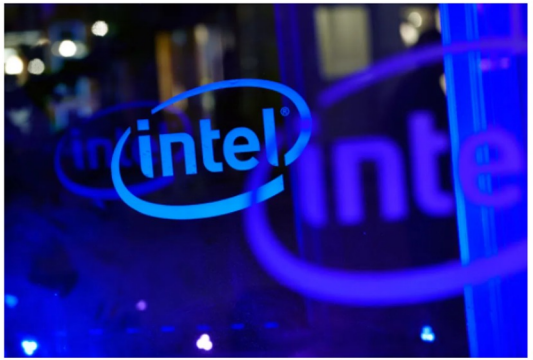
Bugs = Expensive

"Debugging, on average, has grown to consume more than 60% of today's ASIC and SoC verification effort."
-Harry Foster, Mentor Graphics Corporation



Intel does its best to tamp down impact of Spectre and Meltdown in earnings call

Ron Miller @ron_miller / 10:31 AM EST • January 26, 2018 Comment



YOUTUBE · Published December 14

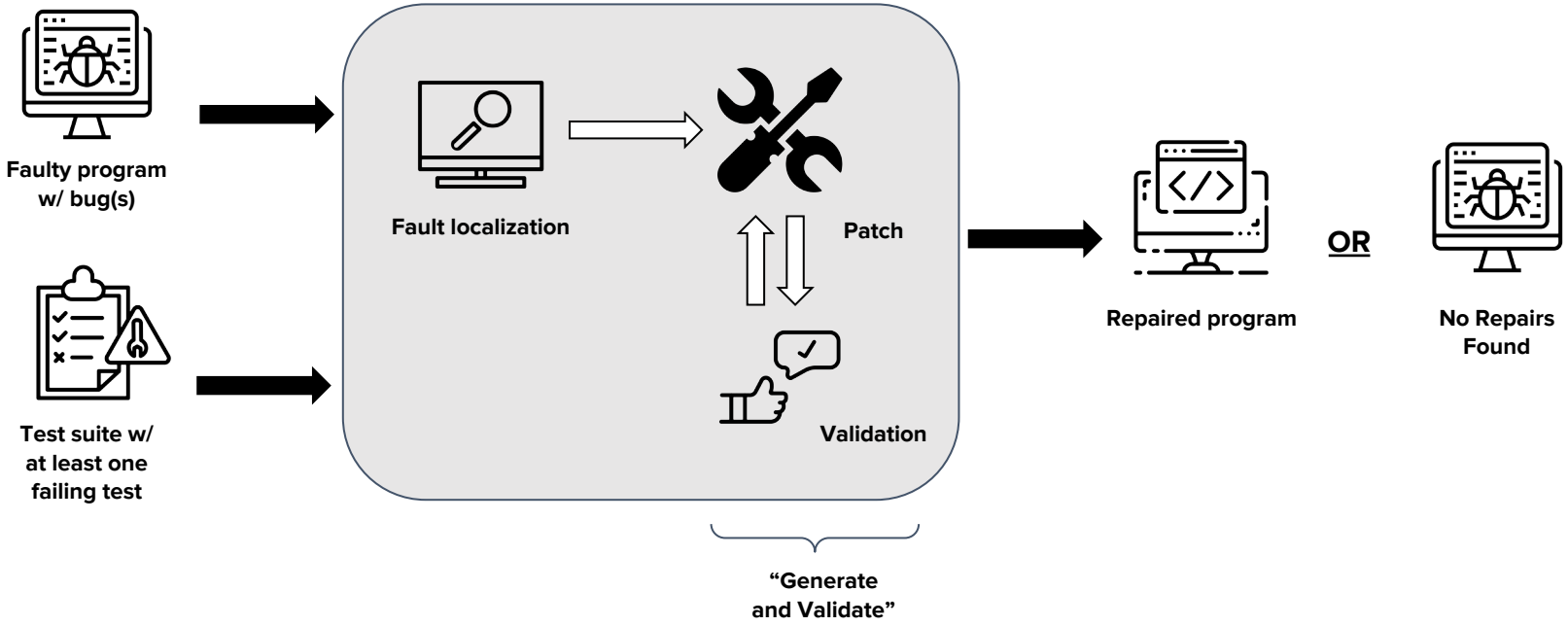
Google lost $1.7M in ad revenue during YouTube outage, expert says

YouTube and other Google services, such as Gmail, suffered outage Monday morning

Amazon's one hour of downtime on Prime Day may have cost it up to $100 million in lost sales

Sean Wolfe Jul 19, 2018, 10:53 AM

A Solution in the Realm of Software: Automated Program Repair (APR)



Problem #1: Software-based APR is not amenable to traditional hardware testbenches

Test suite with two failing tests

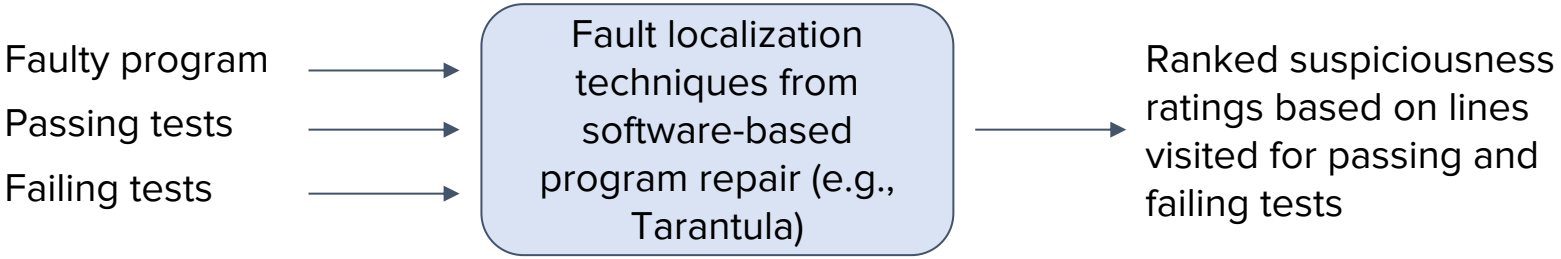
tc0: pass
tc1: pass
tc2: fail
tc3: pass
tc4: fail
tc5: pass
tc6: pass
tc7: pass

→ **Fitness = 0.75** (6 passing, 2 failing tests)

```
Compiler version N-2017.12-SP2-1_Full64; Runtime version N-2017.12-SP2-1_Full64;  Jan 11 11:37 2021
                time,   clk,    reset,  enable, count_out,      overflow_out
                   0,   0,      0,      0,      x,              x
                   5,   1,      0,      0,      x,              x
                                        ...
                 250,   0,      0,      1,      5,              1
                 255,   1,      0,      1,      5,              1
                 256,   1,      0,      1,      6,              1
$finish called from file "first_counter_tb_t3.v", line 70.
$finish at simulation time                  258
```

→ **Fitness = ???**

Problem #2: Fault localization approaches from software-based APR do not scale to hardware



But…

Hardware designs are ***parallel*** in nature!

Introducing: *CirFix*

CirFix: A hardware-design focused automated repair algorithm based on genetic programming (GP)

- Proposes a novel dataflow-based fault localization approach for hardware designs to implicate faulty design code
- Presents a novel approach to guide the search for a hardware design repair using the existing hardware verification process
- *SPOILER:* Fixes hardware defects with a repair rate similar to that of established software-based APR techniques

Fitness Function

- Fitness scores to evaluate candidate repairs
- Testbench ***instrumentation*** to record the values of wires and registers at specified timesteps
- ***Bit-level comparison*** of instrumented wires and registers against expected behavior

Fitness Function: Comparison

Oracle: a developer-provided information for circuit behavior

Simulation output

Oracle

$$sum(S,O) = \sum_{t=0}^{k}\sum_{b=0}^{n} \begin{cases} 1 & (O_{t,b}, S_{t,b}) \in \{(0,0),(1,1)\} \\ \varphi & (O_{t,b}, S_{t,b}) \in \{(x,x),(z,z)\} \\ -1 & (O_{t,b}, S_{t,b}) \in \{(1,0),(0,1)\} \\ -\varphi & (O_{t,b}, S_{t,b}) \in \{(\_,x),(x,\_),(z,\_),(\_,z)\} \end{cases}$$

$$total(S,O) = \sum_{t=0}^{k}\sum_{b=0}^{n} \begin{cases} 1 & (O_{t,b}, S_{t,b}) \in \{(0,0),(1,1),(1,0),(0,1)\} \\ \varphi & (O_{t,b}, S_{t,b}) \in \{(\_,x),(x,\_),(x,x),(z,\_),(\_,z),(z,z)\} \end{cases}$$

$$fitness(S,O) = \begin{cases} 0 & sum(S,O) < 0 \\ \frac{sum(S,O)}{total(S,O)} & sum(S,O) \geq 0 \end{cases}$$

x: uninitialized variable

z: high impedence

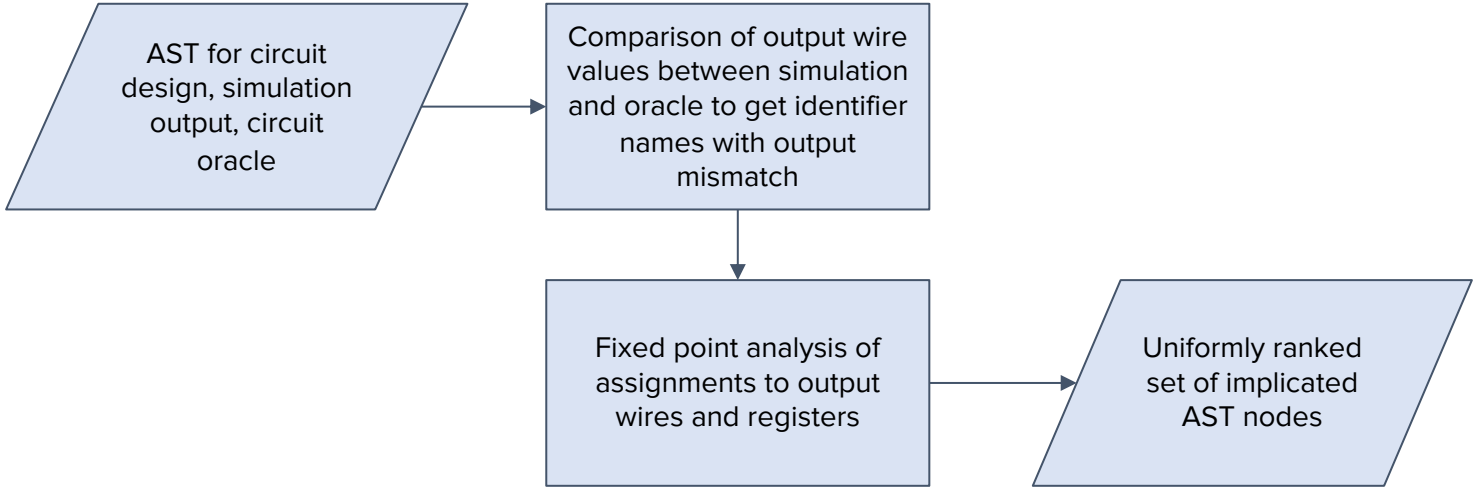
_: bit value of 0 or 1

$S_{t,b}$: b^{th} bit for time t in output

$O_{t,b}$: b^{th} bit for time t in oracle

Fault Localization

Produces a uniformly ranked set of implicated design code for a faulty circuit description



Fixed Point Analysis of Assignments

Input: Faulty circuit design code AST, ast.
Input: Output from design simulation, $S : Time \mapsto Var \mapsto \{0, 1, x, z\}$.
Input: Oracle for circuit behavior, $O : Time \mapsto Var \mapsto \{0, 1, x, z\}$.
Output: Fault localization set, FL.

1: $FL, mismatch \leftarrow \emptyset, \emptyset$
2: $mismatch' \leftarrow$ get_output_mismatch(O, S)
3: **while** $mismatch \neq mismatch'$ **do**
4: $mismatch \leftarrow mismatch \cup mismatch'$
5: **for** $node$ in ast **do**
6: **if** implicated$(node, mismatch)$ **then**
7: $FL \leftarrow FL \cup \{node.id\}$
8: **for** each $child$ of $node$ **do**
9: $FL \leftarrow FL \cup \{child.id\}$
10: **if** type$(child)$ = Identifier and $child.name \notin mismatch$ **then**
11: $mismatch' \leftarrow mismatch' \cup \{child.name\}$
12: **return** FL

Returns a set of wire/register names that have output mismatch

Two ways to be implicated:

- Assignments: assigned variable in the mismatch set
  (e.g., `count <= 1'b1;`)
- Conditionals: conditional includes a mismatched variable
  (e.g., `if(reset==1'b1) count <= 1'b0;`)

More on Methodology in the Paper!

- **Selection**: “choosing parent(s) to produce offspring(s) for the next generation of GP evolution”
- **Repair Operators**: “borrowing code from elsewhere in the parent’s design to produce a child”
- **Repair Templates**: “introducing new design code to the parent to produce a child”
- **Fix Localization**: “guidelines for the APR algorithm to apply edits to design code”

Benchmark Suite of Defect Scenarios

A ***defect scenario*** consists of:

- A Verilog circuit design
- An instrumented testbench for the design
- A developer-provided oracle for circuit behavior
- A design defect for the circuit

Benchmark Suite of Defect Scenarios

A ***defect scenario*** consists of :

- A Verilog circuit design
- An instrumented testbench for the design
- A developer-provided oracle for circuit behavior
- ~~A design defect for the circuit~~
- A ***seeded defect*** by a hardware expert

Benchmark Suite: Hardware Projects

Project	Description	LOC	
decoder_3_to_8	3-to-8 decoder	25	Introductory-level VLSI course projects
counter	4-bit counter with an overflow bit	56	
flip_flop	T-flip flop	16	
fsm_full	Finite state machine	115	
lshift_reg	8-bit left shift register	30	
mux_4_1	4-to-1 multiplexer	19	
i2c	Two-wire, bidirectional serial bus for data exchange	2018	OpenCores projects
sha3	Cryptographic hash function	499	
tate_pairing	Core for running the Tate bilinear pairing algorithm for elliptic curves	2206	
reed_solomon_decoder	Core for Reed-Solomon error correction	4366	
sdram_controller	Synchronous DRAM (SDRAM) memory controller	420	Open-source GitHub project

Benchmark Suite: Defect Seeding

Recruited three hardware experts to seed defects into circuits

Two categories of defects

- Category 1 (i.e., “easy”)
- Category 2 (i.e., “hard”)

32 defect scenarios in benchmark suite

- 19 Category 1 defects
- 13 Category 2 defects

Experimental Setup

RQ #1. What fraction of defect scenarios can CirFix repair?

RQ #2. Does CirFix perform better at Category 1 (“easy”) defects compared to Category 2 (“hard”) defects?

RQ #3. How effective is the CirFix fitness function at guiding the search a repair? (Spoiler: highly effective; more in the paper!)

RQ #4. How sensitive is CirFix to the quality of the information for expected behavior? (Spoiler: not very sensitive; more in the paper!)

RQ #1: Repair Rate for CirFix

CirFix found 21/32 (65.6%) *plausible* repairs, with 16/32 (50%) deemed to be *correct* (i.e., high quality) upon manual inspection

- 2.05 hours average wall-clock time to find a repair

RQ #1: Repair Rate for CirFix

CirFix found 21/32 (65.6%) *plausible* repairs, with 16/32 (**50**%) deemed to be *correct* (i.e., high quality) upon manual inspection

- 2.05 hours average wall-clock time to find a repair

Repair rate comparable to strong results from software-based program repair (e.g., GenProg's **52.4%**, Angelix's **34.1%**)

RQ #2: Performance for Individual Defect Categories

CirFix repaired 12 out of 19 (63.2%) Category 1 (i.e., “easy”) defects and 9 out of 13 (69.2%) Category 2 (i.e., “hard”) defects

- 1.9 hours average wall-clock time to repair Category 1 defects, 2.2 hours average wall-clock time to repair Category 2 defects
- No evidence of statistically significant difference in the average amount of time to find a repair between Category 1 and 2 defects (two-tailed Mann Whitney U test, p = 0.374)

Conclusion

- CirFix: a framework for automatically repairing defects in hardware designs with a 50% repair rate
- Fitness function based on visibility and comparison
- Fault localization approach based on fixed point analysis of assignments
- First publicly available benchmark for a variety of Verilog defects
  - Replication Materials: https://github.com/hammad-a/verilog_repair

Questions?

Feel free to contact Hammad Ahmad (hammada@umich.edu) to start a discussion!