Buggy RTL Hardware Design + Failing Test
Buggy RTL Hardware Design + Failing Test → Human Engineer → Repaired RTL Hardware Design
Automated Program Repair

Buggy RTL Hardware Design + Failing Test

Human Engineer

RepairedRTL Hardware Design

Software + Compute

Repaired?? RTL Hardware Design
Automated Program Repair

Buggy RTL Hardware Design + Failing Test

Human Engineer

Software + Compute

Prior Work:
~50% of repairs introduce new bugs

Repaired RTL Hardware Design
Automated Program Repair

- Buggy RTL Hardware Design
- Failing Test
- Human Engineer
- Software + Compute

Prior Work: minutes - hours

Repaired RTL Hardware Design
Automated Program Repair

- **Buggy RTL Hardware Design**
- **Failing Test**

- **Human Engineer**
- **Software + Compute**

- **Repaired RTL Hardware Design**

Prior Work: minutes — hours
Our RTL-Repair: seconds
module counter(. . .); 
always@ (posedge clock) begin 
    if (reset) begin 
        overflow <= 1'b0; 
        count <= 4'b0; 
    end else if (enable) begin 
        count <= count + 1; 
    end 
    if (count == 'd1) begin 
        overflow <= 1'b1; 
    end 
end 
endmodule
module counter(...);
always@(posedge clock) begin
    if(reset) begin
        overflow <= 1'b0;
        count <= 4'b0;
    end else if(enable) begin
        count <= count + 1;
    end
    if(count == 'd1) begin
        overflow <= 1'b1;
    end
end
endmodule
module counter(...);
always@(posedge clock) begin
  if(reset) begin
    overflow <= 1'b0;
    count <= 4'b0;
  end else if(enable) begin
    count <= count + 1;
  end
  if(count == 'd1) begin
    overflow <= 1'b1;
  end
end
endmodule

<table>
<thead>
<tr>
<th>reset</th>
<th>enable</th>
<th>count</th>
<th>overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
module counter();
always @(posedge clock) begin
    if(reset) begin
        overflow <= 1'b0;
        count <= 4'b0;
    end else if(enable) begin
        count <= count + 1;
    end
    if(count == 'd1) begin
        overflow <= 1'b1;
    end
endmodule

Where do we need to change our code?

<table>
<thead>
<tr>
<th>reset</th>
<th>enable</th>
<th>count</th>
<th>overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
module counter(...);
always@ (posedge clock) begin
    if (reset) begin
        overflow <= 1'b0;
        count <= 4'b0;
    end else if (enable) begin
        count <= count + 1;
    end
    if (count == 'd1) begin
        overflow <= 1'b1;
    end
end
endmodule

After 1s on my laptop:
Add the following:
count <= 4'b0;

<table>
<thead>
<tr>
<th>reset</th>
<th>enable</th>
<th>count</th>
<th>overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
module counter(...);

always@ (posedge clock) begin
    if (reset) begin
        overflow <= 1'b0;
        count <= 4'b0;
    end else if (enable) begin
        count <= count + 1;
    end
    if (count == 'd1) begin
        overflow <= 1'b1;
        count <= 4'b0;
    end
endmodule

<table>
<thead>
<tr>
<th>reset</th>
<th>enable</th>
<th>count</th>
<th>overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
module counter(. . .);
always@/(posedge clock) begin
    if(reset) begin
        overflow <= 1'b0;
        count <= 4'b0;
    end else if(enable) begin
        count <= count + 1;
    end
    if(count == 'd1) begin
        overflow <= 1'b1;
        count <= 4'b0;
    end
endmodule

reset | enable | count | overflow
--- | --- | --- | ---
1 | x | x |
0 | 1 | 0 | 0 |
0 | 1 | 1 | 0 |
0 | 1 | 0 | 1 |
<table>
<thead>
<tr>
<th></th>
<th>RTL-Repair</th>
<th></th>
<th>CirFix [6]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>median</td>
<td>max</td>
</tr>
<tr>
<td>Correct Repairs</td>
<td>16</td>
<td>0.70s</td>
<td>13.17s</td>
</tr>
<tr>
<td>Wrong Repairs</td>
<td>2</td>
<td>0.51s</td>
<td>0.68s</td>
</tr>
<tr>
<td>Cannot Repair</td>
<td>14</td>
<td>5.64s</td>
<td>59.81s</td>
</tr>
<tr>
<td></td>
<td>#</td>
<td>median</td>
<td>max</td>
</tr>
<tr>
<td>------------------</td>
<td>---</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td><strong>Correct Repairs</strong></td>
<td>16</td>
<td>0.70s</td>
<td>13.17s</td>
</tr>
<tr>
<td><strong>Wrong Repairs</strong></td>
<td>2</td>
<td>0.51s</td>
<td>0.68s</td>
</tr>
<tr>
<td><strong>Cannot Repair</strong></td>
<td>14</td>
<td>5.64s</td>
<td>59.81s</td>
</tr>
</tbody>
</table>
RTL-REPAIR: Fast Symbolic Repair of Hardware Design Code

Kevin Laeuffer
laeuffer@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Brandon Fajardo*
brfajardo@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Abhik Ahuja*
ahujaabhik@berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Vighnesh Iyer
vighnesh.iyer@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Borivoje Nikolić
bora@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Koushik Sen
ksen@eecs.berkeley.edu
University of California, Berkeley
Berkeley, CA, USA

Please join us for our full length talk!

Paper PDF
Code on Github