RFUZZ: Coverage-Directed Fuzz Testing of RTL on FPGAs

Kevin Laeufer, Jack Koenig, Donggyu Kim, Jonathan Bachrach and Koushik Sen
University of California, Berkeley
laeufer@cs.berkeley.edu
Dynamic Verification
Dynamic Verification

Design Under Test
Dynamic Verification

Stimuli → Design Under Test
Dynamic Verification

Stimuli ➔ Design Under Test ➔ Coverage
Dynamic Verification

- Stimuli
  - 1) hard-coded

- Design Under Test

- Coverage
Dynamic Verification

Stimuli
1) hard-coded
2) generators (directed random testing)

Design Under Test

Coverage
Dynamic Verification

Stimuli
1) hard-coded
2) generators (directed random testing)

Design Under Test

Coverage
1) automated (e.g. branch, fsm)
Dynamic Verification

Stimuli
1) hard-coded
2) generators (directed random testing)

Design Under Test

Coverage
1) automated (e.g. branch, fsm)
2) annotated (e.g. SV cover)
Dynamic Verification

Stimuli
1) hard-coded
2) generators (directed random testing)

Design Under Test

Coverage
1) automated (e.g. branch, fsm)
2) annotated (e.g. SV cover)

Coverage Report
Interpreted by Verification Engineer
Dynamic Verification

Stimuli

Design Under Test

Coverage
1) automated (e.g. branch, fsm)
2) annotated (e.g. SV cover)

Coverage Report
Interpreted by Verification Engineer
Dynamic Verification

CDG
Coverage
Directed Test Generation

Design Under Test

Coverage
1) automated (e.g. branch, fsm)
2) annotated (e.g. SV cover)

Coverage Report
Interpreted by Verification Engineer
Dynamic Verification

CDG
Coverage Directed Test Generation

Design Under Test

Coverage
1) automated (e.g. branch, fsm)
2) annotated (e.g. SV cover)

Coverage Report
Interpreted by Verification Engineer

Output
Ignored in this work.
We use **Fuzz Testing** to generate stimuli from coverage feedback
Input Definition

Design Under Test

Test Input
Test Input

Input Definition

Design Under Test
Input Definition

Design Under Test

Test Input
Input Definition

Design Under Test

Test Input
Coverage Definition

Functional Coverage
Coverage Definition

Functional Coverage

based on developer intent
Coverage Definition

Functional Coverage

based on developer intent

not available for open source designs
Coverage Definition

Functional Coverage

- based on developer intent

Automatic Coverage

- not available for open source designs
Coverage Definition

**Functional Coverage**
- based on developer intent
- not available for open source designs

**Automatic Coverage**
- used to track test quality in absence of functional coverage
### Coverage Definition

<table>
<thead>
<tr>
<th>Functional Coverage</th>
<th>Automatic Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>based on developer intent</td>
<td>used to track test quality in absence of functional coverage</td>
</tr>
<tr>
<td>not available for open source designs</td>
<td>normally derived from HDL source, not RTL</td>
</tr>
</tbody>
</table>
Coverage Definition

Functional Coverage
- based on developer intent
- not available for open source designs

Automatic Coverage
- used to track test quality in absence of functional coverage
- normally derived from HDL source, not RTL

→ we need an **automatic** coverage metric **based on RTL netlist**
Mux (Control) Toggle Coverage
Mux Toggle Coverage

Toggled?
Mux Toggle Coverage

always @(posedge clk)
begin
  if (a) begin
    a_out = x;
  end else begin
    a_out = f(y,z);
  end
  if (b) begin
    r <= a_out;
  end
end
always @ (posedge clk)
begin
  if (a) begin
    a_out = x;
  end else begin
    a_out = f(y, z);
  end
  if (b) begin
    if (a) begin
      r <= x;
    end else begin
      r <= f(y, z);
    end
  end
end
always @(posedge clk)
begin
  if (a) begin
    a_out = x;
  end else begin
    a_out = f(y, z);
  end
if (b)
begin
  if (a)
  begin
    r <= x;
  end else begin
    r <= f(y, z);
  end
end
end

Background: Coverage-Directed Fuzzing
(a small part of the)
Input Space
Input Definition

Design Under Test

Test Input
Assertion
Violating Input
Input Seed
Intermediate Coverage Goals
Intermediate Coverage Goals
Coverage-Directed Fuzzing: An Example
Fuzzing Example: GCD

Greatest Common Divisor

GCD(a,b)
Fuzzing Example: GCD

```vhdl
io.start_rdy := y === 0.U
when(io.start_en) {
    x := io.start_a
    y := io.start_b
}

when ((x > y) && y /= 0.U) { // swap
    x <= y
    y <= x
}

when ((x <= y) && y /= 0.U) { // subtract
    y <= y - x
}
```
Fuzzing Example: GCD

```vhdl
io.start_rdy := y === 0.U
when(io.start_en) {
    x := io.start_a
    y := io.start_b
}
assume(io.start_en |-> io.start_rdy)

when ((x > y) && y /= 0.U) { // swap
    x <= y
    y <= x
}
when ((x <= y) && y /= 0.U) { // subtract
    y <= y - x
}
```
Fuzzing Example: GCD

\[
\begin{align*}
\text{io.start rdy} & := \text{y} \equiv 0.U \\
\text{when} \{\text{io.start en}\} & \{ \\
& \quad \text{x} := \text{io.start a} \\
& \quad \text{y} := \text{io.start b} \\
& \} \\
\text{assume}(\text{io.start en} \mid \rightarrow \text{io.start rdy})
\end{align*}
\]

\[
\begin{align*}
\text{when} \((\text{x} > \text{y}) \& \& \text{y} \neq 0.U) & \{ \text{ // swap} \\
& \quad \text{x} \leq \text{y} \\
& \quad \text{y} \leq \text{x} \\
& \}
\end{align*}
\]

\[
\begin{align*}
\text{when} \((\text{x} \leq \text{y}) \& \& \text{y} \neq 0.U) & \{ \text{ // subtract} \\
& \quad \text{y} \leq \text{y} - \text{x} \\
& \}
\end{align*}
\]
Fuzzing Example: GCD

```
io.start_rdy := y === 0.U
when io.start_en { 
x := io.start_a
y := io.start_b
}
assume(io.start_en |-> io.start_rdy)

when (x > y) && y /= 0.U) { // swap
  x := y
  y := x
}

when ((x <= y) && y /= 0.U) { // subtract
  y := y - x
}
```
Fuzzing Example: GCD

```plaintext
io.start_rdy := y === 0.U
when {io.start_en} {
    x := io.start_a
    y := io.start_b
}
assume(io.start_en |-> io.start_rdy)

when {(x > y) && y /= 0.U} { // swap
    x <= y
    y <= x
}

when {(x <= y) && y /= 0.U} { // subtract
    y <= y - x
}
```

- `start_a` → `start_rdy`
- `start_b` → `result`
- `start_en` → `result_rdy`
Fuzzing Example: GCD

<table>
<thead>
<tr>
<th>io.start_en</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x &gt; y) &amp;&amp; y != 0.U</td>
<td>-</td>
</tr>
<tr>
<td>(x &lt;= y) &amp;&amp; y != 0.U</td>
<td>-</td>
</tr>
</tbody>
</table>
Fuzzing Example: GCD

Input 0

<table>
<thead>
<tr>
<th></th>
<th>start_a</th>
<th>start_b</th>
<th>start_en</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

io.start_en  -

\((x > y) \land y \neq 0\)  -

\((x \leq y) \land y \neq 0\)  -

Greatest Common Divisor

GCD(a,b)
Fuzzing Example: GCD

Greatest Common Divisor $\text{GCD}(a,b)$

\[
\begin{array}{c|c}
\text{io.start_en} & - \\
(x > y) \&\& y =/= 0.U & - \\
(x \leq y) \&\& y =/= 0.U & - \\
\end{array}
\]

+ 4 more cycles of all zeros!
Fuzzing Example: GCD

\[ \text{GCD}(a, b) \]

\[ 10000000000000000000000000000000 \quad \text{start}_a \]

\[ 00000000000000000000000000000000 \quad \text{start}_b \]

\[ 0 \quad \text{start}_\text{en} \]

| \( \text{io.start}_\text{en} \) | - |
| \((x > y) \land y \neq 0.U\) | - |
| \((x \leq y) \land y \neq 0.U\) | - |
Fuzzing Example: GCD

\[ \text{Greatest Common Divisor } \text{GCD}(a,b) \]

\[
\begin{align*}
010000000000000000000000000000000 & \quad \text{start}_a \\
000000000000000000000000000000000 & \quad \text{start}_b \\
0 & \quad \text{start}_\text{en}
\end{align*}
\]

<table>
<thead>
<tr>
<th>io.start_en</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x &gt; y) &amp;&amp; (y /= 0.U)</td>
<td>-</td>
</tr>
<tr>
<td>(x &lt;= y) &amp;&amp; (y /= 0.U)</td>
<td>-</td>
</tr>
</tbody>
</table>
Fuzzing Example: GCD

\[ \text{Greatest Common Divisor} \ \text{GCD}(a,b) \]

\[
\begin{align*}
00100000000000000000000000000000 & \quad \text{start}_a \\
00000000000000000000000000000000 & \quad \text{start}_b \\
\theta & \quad \text{start}_\text{en}
\end{align*}
\]

<table>
<thead>
<tr>
<th>\text{io.start}_\text{en}</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (x &gt; y) \ &amp;&amp; \ y \neq \theta.\text{U} )</td>
<td>-</td>
</tr>
<tr>
<td>( (x \leq y) \ &amp;&amp; \ y \neq \theta.\text{U} )</td>
<td>-</td>
</tr>
</tbody>
</table>
Fuzzing Example: GCD

\[ \text{Greatest Common Divisor } \text{GCD}(a,b) \]

\[ \begin{array}{c|c}
\text{io.start_en} & - \\
\text{(x > y)} \land \text{y} \neq 0.U & - \\
\text{(x <= y)} \land \text{y} \neq 0.U & - \\
\end{array} \]
Fuzzing Example: GCD

000000000000000000000000000000001 \text{ start}_a

000000000000000000000000000000000 \text{ start}_b

0 \text{ start}_en

<table>
<thead>
<tr>
<th>Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{io.start}_en</td>
<td>-</td>
</tr>
</tbody>
</table>
| \text{x} > \text{y} \land \text{y} 
eq 0.U | - |
| \text{x} \leq \text{y} \land \text{y} 
eq 0.U | - |

Greatest Common Divisor \( \text{GCD}(a,b) \)
Fuzzing Example: GCD

<table>
<thead>
<tr>
<th>Condition</th>
<th>Start_a</th>
<th>Start_b</th>
<th>Start_en</th>
</tr>
</thead>
<tbody>
<tr>
<td>io.start_en</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>((x &gt; y) &amp;&amp; y =/= 0.U)</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>((x &lt;= y) &amp;&amp; y =/= 0.U)</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fuzzing Example: GCD

\[ \text{Greatest Common Divisor} \ GCD(a,b) \]

\[
\begin{array}{c}
00000000000000000000000000000000 \quad \text{start_a} \\
01000000000000000000000000000000 \quad \text{start_b} \\
\theta \quad \text{start_en}
\end{array}
\]

<table>
<thead>
<tr>
<th>io.start_en</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x &gt; y) , &amp;&amp; , y \neq 0.U)</td>
<td>-</td>
</tr>
<tr>
<td>((x \leq y) , &amp;&amp; , y \neq 0.U)</td>
<td>-</td>
</tr>
</tbody>
</table>
Fuzzing Example: GCD

\[
\begin{align*}
00000000000000000000000000000000 & \quad \text{start}_a \\
00100000000000000000000000000000 & \quad \text{start}_b \\
0 & \quad \text{start}_\text{en}
\end{align*}
\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>io.start_en</strong></td>
<td><code>-</code></td>
</tr>
<tr>
<td><code>(x &gt; y) &amp;&amp; y =/= 0.U</code></td>
<td><code>-</code></td>
</tr>
<tr>
<td><code>(x &lt;= y) &amp;&amp; y =/= 0.U</code></td>
<td><code>-</code></td>
</tr>
</tbody>
</table>
Fuzzing Example: GCD

Greatest Common Divisor \( \text{GCD}(a,b) \)

\[
\begin{array}{cccc}
\text{io.start_en} & - \\
(x > y) \land y \neq 0.U & - \\
(x \leq y) \land y \neq 0.U & - \\
\end{array}
\]
Fuzzing Example: GCD

\[
\begin{array}{c}
00000000000000000000000000000000 \quad \text{start}_a \\
00000000000000000000000000000000 \quad \text{start}_b \\
1 \quad \text{start}_\text{en}
\end{array}
\]

io.start_en & - \\
(x > y) && y /= 0.\text{U} & - \\
(x <= y) && y /= 0.\text{U} & -
Fuzzing Example: GCD

Greatest Common Divisor $\text{GCD}(a, b)$

<table>
<thead>
<tr>
<th>io.start_en</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x &gt; y) \land y \neq 0.U$</td>
<td>-</td>
</tr>
<tr>
<td>$(x \leq y) \land y \neq 0.U$</td>
<td>-</td>
</tr>
</tbody>
</table>
Fuzzing Example: GCD

Input 1

<table>
<thead>
<tr>
<th>start_a</th>
<th>start_b</th>
<th>start_en</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

io.start_en | ✓ |

(x > y) && y /= 0.U | - |
(x <= y) && y /= 0.U | - |
Fuzzing Example: GCD

Input 1

<table>
<thead>
<tr>
<th>io.start_en</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x &gt; y) &amp;&amp; y /=/ 0.U</td>
<td>-</td>
</tr>
<tr>
<td>(x &lt;= y) &amp;&amp; y /=/ 0.U</td>
<td>-</td>
</tr>
</tbody>
</table>

Generated by flipping a single bit in Input 0
Fuzzing Example: GCD

Input 2
00000000000000000000000000000000 \( \text{start}_a \)
00000000000000000000000000000011111111 \( \text{start}_b \)
1 \( \text{start}_e_n \)

\( \text{io.start}_e_n \) \( \checkmark \)
\((x > y) \&\& y /= 0.U\) -
\((x <= y) \&\& y /= 0.U\) \( \checkmark \)

Generated by flipping 16 bit on byte offsets in Input 0
Fuzzing Example: GCD

Input 3
10000000000000000000000000000000\quad start_a
00000000000000000000000011111111\quad start_b
1\quad start_en

io.start_en   
(x > y) && y /= 0.U
(x <= y) && y /= 0.U

Generated by flipping a single bit in Input 2
Implementation
Deterministic Test Execution

Inputs

Design Under Test

Coverage
Deterministic Test Execution

Inputs

Design Under Test

Coverage
deterministic
Deterministic Test Execution
Deterministic Test Execution

Test 2

Test 1

Design Under Test

Coverage

Isolate!
Deterministic Test Execution

Test 2

Test 1

Design Under Test

Coverage

Isolate!

Two Problems:
Deterministic Test Execution

Test 2

Test 1

Design Under Test

Coverage

Isolate!

Two Problems:

registers with undefined reset (XXXX)
Deterministic Test Execution

Test 2 → Test 1 → Design Under Test → Coverage

Isolate!

Two Problems:
- registers with undefined reset (XXXX)
- memories
Deterministic Test Execution

Two Problems:
- registers with undefined reset (XXXX)
- memories

Isolate!
Meta Reset

```verilog
reg [31:0] r;

always @(posedge clk) begin
  if (reset) begin
    r <= 32'h1993;
  end else begin
    r <= r_next;
  end
end

(a) Register With Reset
```
Meta Reset

(a) Register With Reset

```verilog
reg [31:0] r;

always @(posedge clk) begin
    if (reset) begin
        r <= 32'h1993;
    end else begin
        r <= r_next;
    end
end
```

(b) Register With MetaReset

```verilog
reg [31:0] r;

always @(posedge clk) begin
    if (metaReset) begin
        r <= 32'h0;
    end else begin
        if (reset) begin
            r <= 32'h1993;
        end else begin
            r <= r_next;
        end
    end
end
```
Meta Reset

(a) Register With Reset

(b) Register With MetaReset
**Meta Reset**

Implemented as a FIRRTL compiler pass.

(a) Register With Reset  
(b) Register With MetaReset
Deterministic Test Execution

Two Problems:
- registers with undefined reset (XXXX)
- memories

Isolate!
Sparse Memories

- Observation: short tests, < 100 cycles
Sparse Memories

- Observation: short tests, < 100 cycles
- Number of memory writes bounded by
  \#WritePorts \times Cycles
Sparse Memories

- Observation: short tests, < 100 cycles
- Number of memory writes bounded by
  \#WritePorts \times \text{Cycles}
- Sparse Memories:
  - use CAM to implement hardware hash table
  - reset in single cycle by setting valid bits to 0
  - FIRRTL compiler pass replaces all memories in the DUT with sufficiently large sparse memory implementation
Implementation

CPU

Fuzzer

Shared Memory

Input Buffer

Coverage Buffer

Verilator

DUT
Implementation
Fully Automated Coverage Instrumentation and Harness Generation

RTL (FIRRTL) → FIRRTL Compiler + Custom Transformation Passes → Test Harness (generated from top module interface)
Results
Results
1.) FPGA Speedup?
# FPGA Emulation Speedup

<table>
<thead>
<tr>
<th></th>
<th>Sodor3Stage</th>
<th>Rocket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of FIRRTL</td>
<td>4k</td>
<td>44k</td>
</tr>
<tr>
<td>Verilator</td>
<td>345 kHz</td>
<td>6.89 kHz</td>
</tr>
<tr>
<td>FPGA*</td>
<td>1.7 MHz</td>
<td>1.46 MHz</td>
</tr>
<tr>
<td>Speedup</td>
<td>4.9x</td>
<td>212x</td>
</tr>
</tbody>
</table>
## FPGA Emulation Speedup

<table>
<thead>
<tr>
<th></th>
<th>Sodor3Stage</th>
<th>Rocket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of FIRRTL</td>
<td>4k</td>
<td>44k</td>
</tr>
<tr>
<td>Verilator</td>
<td>345 kHz</td>
<td>6.89 kHz</td>
</tr>
<tr>
<td>FPGA*</td>
<td>1.7 MHz</td>
<td>1.46 MHz</td>
</tr>
<tr>
<td>Speedup</td>
<td>4.9x</td>
<td>212x</td>
</tr>
</tbody>
</table>

* Takes 2-3h for synthesis + place and route.
Results

1.) FPGA Speedup?
2.) Coverage Improvement?
Coverage Results: Methodology

- Fuzz DUT for 2h on a single AWS vCore
Coverage Results: Methodology

- Fuzz DUT for 2h on a single AWS vCore
- Generate random inputs for 2h on a single AWS vCore
Coverage Results: Methodology

- Fuzz DUT for 2h on a single AWS vCore
- Generate random inputs for 2h on a single AWS vCore
- Repeat experiments 4 times and average results
Coverage Results: Methodology

- Fuzz DUT for 2h on a single AWS vCore
- Generate random inputs for 2h on a single AWS vCore
- Repeat experiments 4 times and average results
- Graph **average mux toggle coverage** as a percentage of the maximum number of muxes in the DUT over time
Coverage Results

**FFT**
- Lines of FIRRTL: 1545
- Mux Cover Points: 195
- Coverage Holes after Fuzzing: 85

**Sodor 3 Stage**
- Lines of FIRRTL: 4021
- Mux Cover Points: 746
- Coverage Holes after Fuzzing: 1-4
Coverage Results

Lines of FIRRTL: 2373
Mux Cover Points: 301
Coverage Holes after Fuzzing: 5 - 61

100%

Random  RFUZZ

Time (s)

I2C

Random  RFUZZ

Time (s)

100%

Package

Lines of FIRRTL: 4046
Mux Cover Points: 323
Coverage Holes after Fuzzing: 7-70
Mutation History
Thank you!

Questions?

Kevin Laeufer
laeufer@cs.berkeley.edu

Reproduce + Extend our Results: github.com/ekiwi/rfuzz