FoREnSiC – An Automatic Debugging Environment for C Programs

Roderick Bloem¹ Rolf Drechsler² Görschwin Fey² Alexander Finder²
Georg Hofferek³ Robert Könighofer¹ Jaan Raik³ Urmas Repinski³ André Sülfow²
forensic@lists.iaik.tugraz.at
¹ Graz University of Technology ²University of Bremen ³Tallinn University of Technology

Abstract

FoREnSiC is short for Formal Repair Environment for Simple C and represents an extensible environment for automatic error detection, localization, and correction in C programs. It implements different debugging methods in a unified way. Currently, a scalable simulation-based back-end, a back-end using symbolic execution, and a formal back-end to verify equivalence between a C program and a hardware design are available. FoREnSiC is designed as an extensible framework. Its infrastructure includes a powerful front-end and interfaces to logic problem solvers and can be reused for implementing new program analysis and debugging methods.

Example

Algorithm in C

```c
unsigned gcd(unsigned u, v){
    unsigned sh = 0, res;
    if(u == 0 || v == 0) return res;
    if(u > v) { u = u - v; } else { v = v - u; }
    while((u | v) & 1) {
        u >>= 1;
        v >>= 1;
    }
    while((u & v) & 1) {
        u = u + v;
        v = v;
    }
    return res;
}
```

Reference in C

```c
unsigned ref(unsigned a, b){
    if(a == 0) return b;
    while((a | b) & 1) {
        a = a + b;
    }
    return a;
}
```

Test Cases

<table>
<thead>
<tr>
<th>Algorithm in C</th>
<th>Reference in C</th>
<th>Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(0,0)</code> = 0</td>
<td><code>(0,0)</code> = 1</td>
<td><code>(0,0)</code> = 3</td>
</tr>
<tr>
<td><code>(0,1)</code> = 1</td>
<td><code>(0,2)</code> = 2</td>
<td><code>-</code> and so on</td>
</tr>
<tr>
<td><code>(1,0)</code> = 1</td>
<td><code>(1,2)</code> = 1</td>
<td><code>-</code> and so on</td>
</tr>
<tr>
<td><code>(1,1)</code> = 1</td>
<td><code>-</code></td>
<td><code>-</code> and so on</td>
</tr>
<tr>
<td><code>(2,1)</code> = 2</td>
<td><code>-</code></td>
<td><code>-</code> and so on</td>
</tr>
<tr>
<td><code>(2,2)</code> = 2</td>
<td><code>-</code></td>
<td><code>-</code> and so on</td>
</tr>
<tr>
<td><code>(2,3)</code> = 2</td>
<td><code>-</code></td>
<td><code>-</code> and so on</td>
</tr>
<tr>
<td><code>(3,3)</code> = 3</td>
<td><code>-</code></td>
<td><code>-</code> and so on</td>
</tr>
</tbody>
</table>

Verilog Implementation

```verilog
module gcd(clk, reset, rdy, i_u, i_v, ret_val);
    input wire clk; input wire reset;
    input wire i_u; input wire i_v;
    output wire rdy;
    output wire ret_val;

    reg rdy;
    reg ret_val;

    reg [31:0] i_u;
    reg [31:0] i_v;

    assign ret_val = 0;
    assign rdy = 0;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;

    assign ret_val = ret_val;
    assign rdy = rdy;