Evaluating Pyrope

A Modern Hardware Description Language

Theodore Sudol (TCNJ), Dr. Jose Renau (UCSC), Blake Skinner (UCSC)

Introduction

- HDLs used to design digital circuits
- Popular HDLs lack modern programming constructs and are inexpressive
- Harder to program complex circuit designs

Pyrope

- Goal: Low-level functionality with high-level expressiveness
- Pipeline programming model
- OOP model with classes, traits and inheritance
- Type inference system for less boilerplate code
- 3 top-level blocks:
  - Stages: describe operations on input data
  - Pipes: connect stages together
  - Classes: define custom data types

Verilog

```verilog
// Booth Multiplication Algorithm
module booth(x, r, en, reset, p, last);
input [7:0] x;
input [7:0] r;
input en;
input reset;
output [15:0] p;
reg [7:0] p;
reg last;
always @ (posedge en or posedge reset)
if (reset) begin
  p <= 0;
end else if (en) begin
  p <= r & 255; // 8x8f
  last <= 0;
repeat (8) begin
  if ((p&1) == 0 & last == 1) begin
    p <= p + (p << 8); // 8x8
    end else if ((p&1) == 1 & last == 0) begin
      p <= p + ((~p) << 8);
      end
  end
  last = p & 1;
  p = p >>> 1;
end
endmodule
```

Pyrope

```pyrope
# Booth Multiplication Algorithm
boothLoop:
  boothAdd
  boothShift
  loop:

boothAdd:
  as
  as
  if
  else if
  if
  end
  end
  +/-

boothShift:
  b = @p & 1
  @p = @p >> 1

pipe:
  m as bits8
  r as bits8
  @p as bits16
  @p as bits8

endmodule
```

Research Goal

- How does Pyrope compare to other HDLs?
  - Verilog: industry standard language
  - Compare expressiveness and verbosity
  - Find awkward syntax
  - Suggest needed language constructs

Method: Use Cases

- Compare two languages by writing programs in both of them
  - Two use cases: Booth Multiplication Algorithm and Elliptic Curve Cryptography
  - Mitigate bias by alternating first language for each program

Results

- Pyrope requires fewer and shorter lines of code
- Pyrope implementations written quicker
- Other tests found Pyrope has < 25% lines of code for complex programs
- Easier for HDL novices to use, especially if familiar with Python or Ruby
- Improved Pyrope syntax by adding stage loop

```
<table>
<thead>
<tr>
<th>Design</th>
<th>Verilog LoC</th>
<th>Pyrope LoC</th>
<th>% Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCD</td>
<td>27</td>
<td>17</td>
<td>37</td>
</tr>
<tr>
<td>Accumulator</td>
<td>13</td>
<td>7</td>
<td>46</td>
</tr>
<tr>
<td>Parity</td>
<td>13</td>
<td>4</td>
<td>69</td>
</tr>
<tr>
<td>Router</td>
<td>178</td>
<td>39</td>
<td>75</td>
</tr>
<tr>
<td>FPU</td>
<td>4017</td>
<td>726</td>
<td>77</td>
</tr>
<tr>
<td>XALU</td>
<td>2901</td>
<td>294</td>
<td>90</td>
</tr>
</tbody>
</table>
```

Future Work

- Pyrope is still under development
- More use cases to continue evaluation as language evolves
- More complex and synthesizable tests

Figure 1: Pyrope vs. Verilog – Booth Multiplication Algorithm.
Above: Pyrope implementation.
Right: Verilog implementation.

Figure 3: A comparison of various digital circuits in Verilog and Pyrope. Sizes are measured in lines of code (LoC).

Figure 2: Booth Multiplier circuit
Source: COA Virtual Lab