Evaluating Pyrope

A Modern Hardware Description Language

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

Baskin Engineering
UC Santa Cruz

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

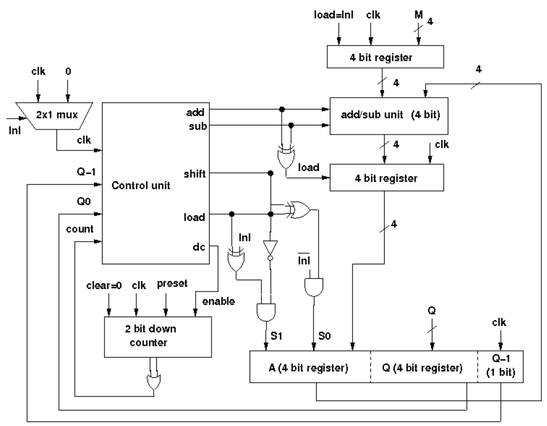

Figure 2: Booth Multiplier circuit
Source: COA Virtual Lab

Verilog

```
// Booth Multiplication Algorithm

module booth(m, r, en, reset,  p, last);

input [7:0] m;
input [7:0] r;
input en;
input reset;

output [15:0] p;

reg [15:0] p;
reg last;

always @ (posedge en or posedge reset)
  if (reset) begin
    p <= 0;
  end else if (en) begin
    p <= r & 255; // 0xff
    last <= 0;
    repeat (8) begin
      if (p&1 == 0 && last == 1) begin
        p <= p + (m << 8);
      end else if ((p&1) == 1 && last == 0) begin
        p <= p + ((-m) << 8);
      end
      last = p & 1;
      p = p >>> 1;
    end
  end
endmodule
```

Pyrope

```
# Booth Multiplication Algorithm

stage boothAdd:
  reset: @P = r

  if (P&1, b) == (0,1):
    @P += (m << 8)
  elif (P&1, b) == (1, 0):
    @P += (-m << 8)

stage boothShift:
  b = @P & 1
  @P = @P >> 1

pipe booth:
  m as bits:8
  r as bits:8
  @P as bits:16
  b as bits:8

  loop boothLoop:
    boothAdd -> boothShift
  boothLoop(8)
```

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

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

Design	Verilog LoC	Pyrope LoC	% Reduction
GCD	27	17	37
Accumulator	13	7	46
Parity	13	4	69
Router	178	39	75
FPU	4017	726	77
xALU	2901	294	90

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

Future Work

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