Pyrope: A Nicer Jewel

A Comparison and Analysis of Hardware Description Languages

Rashad Kayed
University of California, Santa Cruz
Micro Architecture Santa Cruz (MASC)
Faculty Advisor: Jose Renau
Graduate Advisor: Haven Skinner

Abstract— **Pyrope is a new Hardware Description Language that is based on scripting languages and uses Ruby like structure and aspects. The goal for Pyrope is to maintain simple programming methods yet implement high-level capabilities. For example, Pyrope has the ability to design a pipeline structure on its own so that a user has fewer aspects to manage. It is important to build test cases in both Verilog and Pyrope, then analyze and compare the results. Once completed, Pyrope will be a user-friendly Hardware Description Language with outstanding capabilities and possibilities, such as less required code to complete a project.**

Keywords—Hardware Description Language;

I. INTRODUCTION

Hardware Description Languages, also known as HDL's, are wideley used and esssential in the computer engineering industry today. A hardware Description Language is used in electronics to describe a circuit, so that a circuit can be tested with respect to its design, planned funtionality, and operations [1]. Most HDL's in the industry today are outdated and have been around for many decades. Even though these languages may do the job, they lack essential features that are in other languages also used in the industry.

Computer Engineers have many tasks when it comes to prjoects; everything from coming up with an idea to a problem, to collabarating to seek possible ways to solve the issue, and writing the actual code that will get the project done. However, having for example to introudce and initialize each variable you may have to use before starting your actual code, can be a hastle, and should not be an aspect of programing to think twice about.

Today the two most common Hardware Description Languages are VHDL and Verilog. Verilog was first introdced in 1984, and has been developing gradually since it was first used in 1985 [2]. Verilog is similar to the C language, and is a low level hardware language. It is most commonly used when designing and implementing digital circuits at the register transfer level. Verilog is also important for analog circuits and mixed signal circuits. The first major change to Verilog was in 1986, which was gate level simulation, known as the "XL algorithm". Verilog allowed users to model at higher levels of abstraction [2]. In 1988 *Synopsys* made a breakthrough allowing Verilog to be used as an input language, allowing for the top-down coding methodology to be correctly implemented. Finally, after efforts from different parties, Verilog became and IEEE standard in 1995 [2].

Besides these few changes over the decades, not much has changed with Verilog, nor have any major advancements been made. Which prompted for work to be done on introducing a new Hardware Description Language by the name of Pyrope. Pyrope makes simple but important improvements that can make a huge difference in electronics. The focus of our research was to build test cases of code in both Verilog and Pyrope and analyze the outcome of our work. The work on this project helped further support our hypothesis that Pyrope is a more efficient language overall, and the specifics of this will be discussed further in this report.

II. HARDWARE DESCRIPTION LANGUAGES

A hardware Description Language is used in electronics to describe a circuit, so that a circuit can be tested with respect to its design, planned funtionality, and operations [1]. A Hardware Description Language can be used to describe something like a microprocessor or a flip flop switch.

The most common Hardware Description Languages are VHDL and Verilog. These two languages are used in todays industry to compliment projects in electronics and other engineering. Verilog is a language common to C since it uses similar design elements and behaviours. Verilog is most commonly used when designing and implementing digital circuits at the register transfer level.

Pyrope uses some Ruby language aspects, so it was named Pyrope after another similar gemstone. Pyrope creates a simpler language for digital architecture by implementing programming constructs. Pyrope's purpose is to maintain the functionality of low-level Verilog code, yet also implement a highly expressive language with abstraction capabilities [5].

Our work focused on building test cases in both Verilog and Pyrope languages. The point of the test cases is to write the

same code in both Verilog and Pyrope, so that it can be compared and analyzed. The physical coding in the two languages helped us see the difficulties in both languages and come up with suggestions to make Pyrope a more advanced and beneficial Hardware Description Language. One of the test cases built and studied was a booth multiplier.

III. Booth Multiplication Algorithm

Booth's algorithm can be implemented by constantly using unsigned binary addition and performign shifts to the values. First, asign the variable m to be the multiplicand and r to be the multiplier. The variable A will be the value of m, and the remaining leftmost bits are to be filled with zeros. The variable S will hold the value of negative m, and the reamining bits are to be filled in with zeros. For the variable P fill in the most significant bits with zeros and then add the value of r to the end of this [4]. The variable y holds the number of bits in r.

Now to start operating using the algorithm, figure out what the two rightmost bits of P are. If they are 01, compute the value of P+A and ignore overflow. If the rightmost bits of P are 10, compute the value of P+S, and ignore overflow. If the rightmost bits are 00 or 11, do not compute anything and move on [4]. After this arithmetically shift the value you computed for P and change the value of P to now be this new value. Now repeat the computation of P based on the value of the rightmost bits, and shift after each computation 'y' times. Last drop the least significant bit from P and this is your product of m and r.

A. Verilog

In the Verilog implementation of the booth multiplication algorithm you begin by defining and allocating bits to the variables you will use. You then make your mask, which you will use to multiply to your 'P'. Depending on the product you obtain you will choose an arithmetic function to perform. For example, if your p mutiplied with your mask produces the rightmost bits to be '10' you add S to P, and you shift P. After going through the loop up to 'y' times, you shift one final time to obtain the final answer.

```
always @ (posedge en, posedge reset)
for(i=0; i<4; i=i+1) begin
  mask='b000000011;
  if(p&mask == 01) begin
    p=p+a;
    p=p>>1;
  end
    else if(p&mask == 10) begin
      p=p+s;
      p=p>>1;
    end
      else if(p*mask == 00) begin
        p=p;
        p=p>>1;
      end
    else begin
      p=p;
      p=p>>1;
    end
```

B. Pipeline Structure

In Pyrope the code is split into different stages. This is a feature of Pyrope that makes it an easier language to code in; you are able to divide the different functions into 'chunks' called stages. These stages are a representation of the pipeline structure. The blocks are a representation of the logic behind flops or latches. In the pipeline structure as seen in *Fig. 1*, there exists a cloud which represents the combinational logic, meaning logic based solely on the inpute given. Also, there are registers in the pipeline structure which store the data computed or inputed from previous logic. From these registers you are able to reuse the information in the next cloud of computational logic. This is one of the main improvements of Pyrope, because when coding in Pyrope a user does not need to keep track of all their inputs and outputs [5]. The Pyrope stage structure implements this feature for the user.

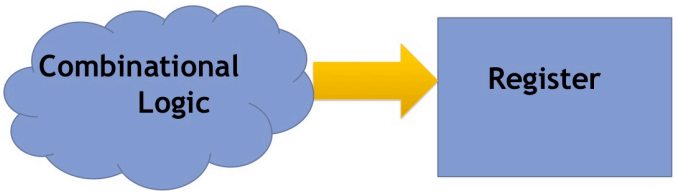

Fig. 1: The pipeline structure, containg a cloud of computational logic, and a register to store output from input.

C. Pyrope

Pyrope offers many advancements and changes to coding with a Hardware Description Language. First, it implements the pipeline structure on its own like mentioned above also it makes using a Hardware Description Language easier and neater. It is easier to spot and organize your code by separating your code into stages. In the first stage of code in the Pyrope implemenation of the Booth Algorithm there is a stage called add. In this stage we begin by declaring our variable P with an ampersand. We then multiply our mask to P and check if the product is either a '01' or a '10', depending on that output you add accordingly. In the next stage named shift, you shift P by one place, since this applies to more than one case having the shift operation in its own block, it makes it easier to access whenever needed.

The next stage, 'drop', is used to drop the last bit in P. The pipe together function is basically matching and inializing the variables to their valuables for the entirity of the program. And the "repeat 8", is a loop that makes the program loop through these stages eight times. Lastly, the program is ended with

arrows linking the different stages together in the appropriate order they are to be referenced.

```
stage add
@p=@p&mask
if(p&mask) == (01)
@p=@p+a
elif (p&mask) == (10)
@p=@p+s'

stage shift
@p=@p>>>1

stage drop
@p=@p>>>1

pipe together
m as bits:8
s as bits:8
p as bits:9
mask as 'b001

repeat 8;
add -> shift ->drop
```

IV. Results

After careful research through reading and building test cases it is evident that Pyrope is a more efficent language than Verilog. Pyrope allows a user Global Type inference, where one can iniliaze their variables somewhere in their code and have the values of those variables saved and accessible. Pyrope was also found to reduce boilerpoint code, which is all the needed declarations and inilizations before one starts to program. For example, in the counter example in *Fig. 2*, all lines of code present can be eliminated or significantly reduced if coding the same example in Pyrope.

```
module counter(en, clk, reset, amt, x);
input en, clk, reset;
input [3:0] amt;
output [7:0] x;
reg [7:0] x;
```

Fig. 2: Formalities such as the code in the box above are not needed in Pyrope, most variables and initilization of their values are implicit.

In Pyrope the two lines of code that would represent a mostly fully functional counter are:

```
@Counter = 8h0
@Counter = @Counter + Amt if En
```

Since there is a reduction to boilerpoint code, as shown in the previous examples, there obviously will be a change in the amount of code between the two languages. We found and confirmed through our research that coding in Pyrope offers 20-40% less lines of code then coding the same example in Verilog[5]. The following table *Fig. 3* displays different coding exmaples performed in both Verilog and Pyrope with their corresponding lines of code, and percent reduction between the two languages.

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

Fig. 3: Soucre: Jose Renau and Haven Skinner

Most importantly the pipeline structure is a huge improvement in the field of Hardware Description Languages, this is because of the impementation of stages. If stages are able to be put into affect and used properly the use of buses will not be neccessary, busses being the arrays of bits. Currently, Verilog has a user manually define constants and there is no support for type checking[5]. In pyrope the user is able to express their code in simpler ways, and functions such as a clock and reset are implicit.

	Verilog/VHDL	System Verilog	Chisel	Bluespec	Pyrope
Expressiveness	Low	Moderate	Moderate	Moderate	High
Type Inference	No	No	Local	Local	Global
Field Inference	No	No	No	No	Yes
Implicit Clock	No	No	Yes	Yes	Yes
Implicit Reset	No	No	No	No	Yes
Implicit Handshake	No	No	No	No	Yes

Fig. 4: This table compares certain features of Hardware Description Languages, between the five specified languages in the table[5].

LESSONS LEARNED

Using Verilog and Pyrope to code and build test cases for our research purposes had both its pros and cons. To start with, Verilog is a Hardware Description Language that has been around for decades, hence, it is well documented. You can easily find sources for learning about Verilog and its specific attributes by reading a book or searching online. Verilog is similar to C, so it is not too hard to catch the rhythm and style of coding in it. On the other hand, Pyrope is a new language still in the building phase, so it does not have much sources of reference to learn more from. This for me was the hardest aspect of using Pyrope, not having much to refer to. Also, in Pyrope it is hard to get accustomed to all the syntax, such as shift operators and dealing with overflow.

V. CONCLUSION

In our research we sought to compare the two languages Verilog and Pyrope. Pyrope is a new Hardware Description Language that aims to offer more user friendly features and aims to bettering the process of using a Hardware Description Language. Our main goals were to build test cases of the same example using the two languages. This would aid us in determining the difficulties when coding in Pyrope rather than Verilog. Also, by becoming familiar with the two languages we would be able to suggest improvements that would better the language. Some of the recommendations that were thought of during our work was as simple as fixing syntax for overflow in the code, and suggesting a VIM compatability highlighter.

ACKNOWLEDGEMENT

This work was supported by UCSC's SURF-IT 2013, funded by the National Science Foundation. I want to acknowledge Professor Jose Renau and Haven Skinner who guided the research and offered support throughout the program. Also, I would like to thank Matthew Guthaus and Colt Hangen and all other faculty that aided in making this program a success.

REFERENCES

[1] http://www.wordiq.com/definition/Hardware_description_language 2010
[2] http://vol.verilog.com/VOL/main.htm 2002
[3] http://www.verilog.com/index.html 2012
[4] http://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm
[5] J. Renau, H. Skinner, "Pyrope", University of California, Santa Cruz, unpublished