Are Modern Hardware Description Languages Ready for FPGA Design?

Martin Schoeberl

Department of Applied Mathematics and Computer Science, Technical University of Denmark

masca@imm.dtu.dk

David Broman

University of California, Berkeley and Linköping University

broman@eecs.berkeley.edu

Addition comments by Christohper L. Felton

cfelton@ieee.org

Abstract

Martin: This is just a first, way too long abstract. Since decades we use VHDL and Verilog as the main entry languages for hardware design. However, those are old languages originally not designed for synthesize. In the mean time the software community has made big improvements in programming languages and increased the productivity. Can we leverage some of the development in programming languages for hardware description? Several new hardware description languages (HDL), based on software programming languages, are proposed. In this paper we evaluate these new HDLs and assess if they are ready for prime time. Describe what we did find out

I. General

Do some exploration of user base and available source code on all languages.

How many *background* knowledge on programming language do we need for the new HDL?

Can it be a language to teach 1st and 2nd semester EE students to learn digital electronics? Even without programming background? We now struggle with VHDL.

Languages to look at:

http://tomahawkins.org/ two languages at the bottom

- MyHDL
- Chisel
- Lava
- Gezel
- JHDL
- Confluence
- HDCamel

II. Introduction

Verilog and VHDL has served us decades as the main languages for designing digital systems. However, both languages have a large legacy and might not be the most efficient languages to (1) build large digital systems and (2) to learn hardware design.

In the software domain there has been a boom in new programming languages in the last years. Programmers are more flexible now to change programming languages when the job to been done is simplified. Within the hardware design community a new HDL has a hard time to get established. As there is a way smaller user base simulation and synthesize tools vendors are reluctant to support a new HDL.

However, in recent year there have ben new approaches to use modern languages as a basis for hardware design. The HDL is *embedded* in those new languages. The tool supporting this HDLs usually provide support for simulation. For synthesize of hardware VHDL or Verilog is generated. Therefore, the *old* HDLs serve as an intermediate language to have the full support of synthesize tools.

This paper compares modern hardware description languages and assesses if they are mature and useful for FPGA designs.

The question is: is hardware design more productive with those new languages? Are these languages mature enough for real world designs? In this paper we explore several new HDLs in the context of hardware design with FPGAs. When switching to a new design methodology one wants to minimize risk and dependency on a a singe vendor. Open source projects fulfill the role of risk minimization. In the worst case one can hire someone to fix a bug in the HDL. Therefore, we restrict our language survey to open-source projects.

A. Assessment Criteria

In synchronous design the two basic building blocks are combinational (asynchronous) logic and flip-flop based registers. Larger on-chip memories are not built out of registers, but are considered another building block. This building blocks are also reflected in current FPGAs: lookup tables (LUT) are used to implement combinational functions, dedicated flip-flops the registers, on-chip memories support larger storage requirements, and dedicated hard macros (with MAC operations) support efficient implementation of DSP algorithms.

This results in following questions:

- Is it clear what is combinational logic and what are registers?
- Can FPGA block RAMs be instantiated? Can ROMs be used?
- Can the DSP blocks be used?

TODO: Write some text around the questions as motivation for them

Further questions to be answered:

- Is simulation in the HDL possible?

- Is there a clear line between the synthsizable and not synthsizable parts of the language?
- Can it be used in a first semester digital electronics course?
- What about programming in the large? Are name spaces supported?
- Quality of generated code: hardware resources and maximum clock frequency
- Mixed language (with VHDL and/or Verilog) usage?
- Size of the user group and support
- Active development

Missing and hard to assess: productivity, easy to maintain.

III. Getting Started

This is a first tour through the languages.

A. MyHDL

Starting with MyHDL was a pleasure. Having Python already installed, the installation of MyHDL was a matter of minutes. Having been exposed to Python for about one day before, the first examples from the MyHDL manual [**?**] where easy as well. However, jumping ahead and trying to generate some real VHDL code and programming an FPGA with the HW version of *Hello World* took some hours struggling with cryptic error messages. The author is used to statically typed languages and compile checks, where Pythons dynamic typing and interpreting execution model reveals program and even typing errors only at runtime needs some adaption of the authors programming mindset and habits.

The development approach in MyHDL is to test and verify the hardware within Python first before generating VHDL or Verilog. Therefore, basic Python language knowledge is needed for MyHDL. In fact MyHDL is less a language in its own, but a Python package.

TODO: Maybe this goes into a Discussion section at the end

The question is if a dynamic typed language, where a variable can change the type at runtime, the best base for HW synthesis is. In the resulting Verilog/VHDL code and in the hardware the type of a signal or a register is statically fixed. Type errors in Python are only detected at runtime by extensive testing. In a statically typed language the compiler finds all type errors at compile time.

***Chris**:* This is a common topic for dynamic languages like Python, strong typing vs. dynamic typing. Python has typing, you can't do $"me" + 8$. The subset of the HDL that explicitly describes the HDL to be converted has to be statically typed. The code is generated before ever running! The types have be be defined. You have this interesting mix of the embedded HDL. As you pointed out, the converter is not a compiler, it does not do (or

intended) to do a thorough job of static analysis. But the error messages are continuously improved by the myhdl community

TODO: Comment: Python is easy going with dynamic typing and test benches run. However, when converting to VHDL or Verilog, types (i.e., lengths of bit vectors) need to be specified. As a result after high level simulation more work is needed to have synthesizable code.

***Chris**:* Yes, myhdl is very similar to existing HDLs, the converter does close to a one-to-one translation from myhld to Verilog/VHDL. But myhdl pushes the abstraction level as high as possible. Example $intbv$ can be defined with min and max vs. the number of bits. Long, time digital engineers we often think in bits but when defining the min and max the intent is much clearer for anyone else reading the code in the future!!!

MyHDL takes a very software oriented approach to hardware design. The emphasis is on unit test, which is a new a interesting approach in hardware design. However, teaching digital electronic with VHDL showed the issues of students thinking too much in the term of a *program* instead of the hardware structure. We think a HDL shall describe the actual structure and not an algorithm in form of a software program.

***Chris**:* software oriented process approach. I totally disagree based on years of teaching HDL as well. Students need to understand what a complex digital system is and how the HDL works. I have not run into many students that do not quickly grasp the concurrent vs. sequential parts of the language. How do you resolve this subjective point? You would need a controlled study, which is very difficult since digital hardware design is a small subset of Eng. I do agree, highly concurrency vs. globally procedural needs to be emphasized when teaching new HDL'ers. This is how many new languages are evolving to support massive parallelism.

- Very simple to download install
- Python needs to be learned, some VHDL/Verilog knowledge is still needed

Installed on Python 2.7. Does it need a 2.x Python or would it run on 3.x as well?

***Chris**:* it requires greater than 2.6 (for the 2.x versions). Limited testing on 3.x.

How large is the user base on MyHDL? Any open-source projects?

***Chris**:* This is hard to gauge. There are many users that come along and implement a project or use for work. But might not revisit for a couple years, the nature of their work. They only implement new HDL every couple years. Overall the active user base is smallish, around a dozen or so

see http://thread.gmane.org/gmane.comp.python.myhdl/2701

1) Remarks: Why does the manual show code for a simple mix that is not convertible to VHDL?

***Chris**:* ?? some pointers to these examples would be useful. In the manual it does differentiate between modeling (which the myhdl author and contributors think is important) and conversion. Modeling there are examples that are not intended to be converted. All intended conversion examples should convert. Also see the "myhdl by example" wiki page on myhdl.org

This is not the way we would like the process for synchronous logic:

```
FOO_HDL: process (clk, reset) is
```

```
begin
    if (reset = '1') then
        sig <= '0';
    elsif rising_edge(clk)or falling_edge(reset) then
        sig <= to_std_logic((not to_boolean(sig)));
    end if;
end process FOO_HDL;
```

The MyHDL manual introduces too early, too many non-synthesizable constructs. They might be good for test benches, but that should be stated more explicit.

***Chris**:* The author and contributors of myhdl believe the modeling to be one of the important features of myhdl. Not only implementation (conversion). Many developers build models and do trade-off studies before ever implementing. Have a process to work through a design (vs. simply an implementation) the myhdl community believes is important and modeling is part of that. This is where it gets difficult in small examples. Obviously, in small academic examples modeling before had etc.

The following is an example of creating a reference model first before moving onto the implementation. http://www.programmablepla

$2438doc_id = 254265$

Positive is conversion to VHDL and Verilog.

B. Chisel

Chisel is a HDL developed at the University of California, Berkeley [**?**]. Chisel is embedded in Scala.

As the language is a research vehicle and mainly used internally, getting started just with the tutorial is harder than getting started with MyHDL. Besides learning Chisel, some knowledge of Scala is needed. Furthermore, the tool flow for Chisel is based on the build tool sbt, which itself needs to be learned.

The tutorial for Chisel is already a complex set of exercises, solution, and automation for building those exercises. In our opinion this starts with a too steep learning course. Therefore, we have cut down the tutorial to the minimum of a hardware *Hello World* program: a blinking LED on an FPGA board.[1]

TODO: The tutorial starts too complex.

***Martin**:* Are enums missing? Does the bit width always need to be known?

***Martin**:* Check if asynchronous or synchronous reset is used.

***Martin**:* Looks like synchronous reset is used. That's bad :-(

1) Issues, Questions: The automatic inference of the correct bit length can lead to surprising results. "val r1 = Reg(resetVal = UFix(0, size))" describes a register with size bits, but the expression "val r1 = Reg(UFix(0, size))" results in a single bit register.

[1]This Chisel Hello World introduction is available as part of our HDL comparison code at: https://github.com/schoeberl/comphdl/tree/master/hello-chisel

2) FlexPRET/Chisel Questions: Why are the pipeline registers individual gals per filed? Why not a Bundle, which could also be used for component IO?

C. Lava

Lava is a HDL based on Haskell [**?**]. The first version described is focused on verification and does not yet support description of sequential hardware. *TODO: Assume that this is all gone and there are some more papers.*

Martin: Looks like this Lava thing is tight with Xilinx. VHDL code generation is also mentioned.

IV. Language Assessments

In this section we will assess which languages fulfill the criteria proposed in the introduction and answer the questions.

A. Is it clear what is combinational logic and what are registers?

TODO: Check Verilog

In VHDL there is no clear distinction between combinational logic, flip-flop registers, and memories in the language. Registers (and memories) are inferred from *code patterns*. Furthermore, small errors can result in unintended latches generated in processes that are intended to be combinational.

MyHDL has no distinct types for register of memory. However, it uses annotations (so called decorators in Python) to mark a function representing combinational logic with `@always_comb` and a function where the output is stored in registers with `@always(clk.posedge)`. We consider this as an enhancement related to VHDL/Verilog *TODO: Check Verilog*.

B. Documentation and Tutorials

The MyHDL distribution contains a mental [**?**] that includes a tutorial.

C. Active Community

It is important that a new language has a active developer group that is able to fix errors and incorporate changes when new (FPGA) technology becomes available. Furthermore, an active forum for the language users is beneficial for getting started.

MyHDL is developed and maintained by Jan Decaluwe with the help of Christopher Felton.

V. Notes (from iPad/Dropbox)

MyHDL

Having min and max values that cannot be implemented so in HW is strange. I would prefer just signed/unsigned with a power of 2 range.

MyHDL is not a new language, but a python package. Python is the language. If a HW description is embedded in a full blown general purpose language with a lot of libraries, how easy is it to see the boundaries of what is synthesizable hardware, what is test benches, unit tests, and even generator code?

There is no indication is parameters are input or output signals.

Testing looks convenient with all the Python support.

Are signal records/structures possible?

A dynamic (scripting) language depends very much on unit tests. Even typos are only discovered when the code is executed. Static typed languages (such as Java and Scala) can catch quite a lot of bugs and typos during compilation.

General

Is tri state supported?

Shall I look into C based HDLs?

SystemC?

SystemVerilog

VI. Further Notes and Reading Material

MyHDL news group archive: http://blog.gmane.org/gmane.comp.python.myhdl

VII. A New Language

This is a start, which shall go into it's own paper, of a new language. Name is needed, but the current acronym is SHDL.

SHDL shall be a simple, minimalistic language to generate synthesizable hardware via an intermediate language, such as VHDL. SHDL shall be simple to learn and it shall be clear which constructs are synthesizable. Maybe all constructs are synthesizable and a simulation needs another language to implement the test bench.

One suggestion is that we implement the new DSL as an embedded DSL in Modelyze. The important thing is, however, that this DSL is *exclusively embedded*, meaning that it does not derive more functionality from the host language (Modelyze in this case) than necessary or wanted. For instance, our new HDL should perhaps require all translations to VHDL to terminate and only allow static typing. We may use the term *fencing* for describing how the embedded DSL is separated from the host language.

A. Short Notes

Martin: I like the x.next notion from MyHDL for register values as it shows that the value will change on the next clock tick and is not yet available. However, MyHDL uses this for combinational signals as well, which is not so good.

Most languages handle as primary target circuit simulation and synthesis as after thought. This often leads to a language, where it is not clear which constructs can be synthesized and which not. However, we shall approach the circuit design with *Synthesize first* to have a clear notion of synthesizable constructions in the language.

B. Concepts

We should make explicit and clear fundamental concepts

- Component abstraction
- Component instances
- Wiring between components
- Registers
- Logic
- Parameterization

C. Simple circuits with logic only

```
comp Xor2()
  in i1, i2 : bool
  out o1 : bool
  o1 = i1 xor i2
end

comp Not1()
  in i1 : bool
  out o1 : bool
  o1 = !i1
end

comp SimpleCircuit()
  in  a : bool[3]
  out even : bool = unit3.o1

  def unit1 = Xor2()
  unit1.i1 = a[0]

  def unit2 = Xor2()
  unit2.i1 = unit1.o1
  unit2.i1 = a[2]

  def unit3 = Not1()
  unit3.i1 = unit2.o1
end
```

We use def to define a local name. In the above case is it used for naming component instances.

```
type R1 = {
  a:int
```

```
  b:int
}

type R2 = {
  a:int
  b:int
}

type R3 = {
  a:int
  b:int
}

comp A()
  out o2:R2
  out o3:R3
end

comp B()
  in i4:R4
  in i3:R3
  out o1:R1
end

comp C
  in i1:R1
  in i2:R2
  out o4:R4
end

comp Circuit
  def a = A()
  def b = B()
  def c = C()
  b.i3 = a.o3
  b.i4 = c.04
  c.i1 = b.o1
  c.i2 = a.o2
end


comp Counter
  in clear:bool
```

```
  reg cnt:int4 = a
  def a = if !clear then cnt + 1 else 0
  out v:int4 = cnt
end
```

Use term single assignment

VIII. Conclusion

IX. Paper Plan

Comparison to FPL 22 March

Extension of comparison to TII or TCAD

New language: Euromicro on DSD 11 March, ESWEEK 29 March, although best fit would be DAC and DATE

Embedded language without seeing the host: some program languages conference