White Paper - Investigate the high-level HDL Chisel

Florian Heilmann, Christian Brugger, Norbert Wehn
Microelectronics Research Group, University Kaiserslautern
Kaiserslautern, Germany
fheilman@rhrk.uni-kl.de, brugger@eit.uni-kl.de, wehn@eit.uni-kl.de

***Abstract*— Chisel (Constructing Hardware in a Scala embedded language) is a new programming language, which embedded in Scala, used for hardware synthesis. It aims to increase productivity when creating hardware by enabling designers to use features present in higher level programming languages to build complex hardware blocks. In this paper, the most advertised features of Chisel are investigated and compared to their VHDL counterparts, if present. Afterwards, the authors' opinion if a switch to Chisel is worth considering is presented. Additionally, results from a related case study on Chisel are briefly summarized. The author concludes that, while Chisel has promising features, it is not yet ready for use in the industry.**

Keywords—Hardware design; Chisel; VHDL; HDL

I. INTRODUCTION

Traditional HDLs were originally conceived to be hardware simulation languages, not hardware synthesis languages. Although more recent revisions and standards exist, the versions supported by the most vendors are quite old (VHDL: 1993, Verilog: 2001). While they are still almost ubiquitously used in the industry, their age and original intention puts them behind current high level languages in terms of productivity and flexibility. Chisel aims to provide a language embedded in the Scala programming language to provide these new features to hardware designers while, at the same time, speeding up development by being able to generate fast simulations of the design. This paper evaluates some of the advertised features of Chisel, comparing their advantages in productivity and code size to their VHDL counterparts, if present. Additionally, this paper briefly discusses some difficulties with Chisel in its current state based on a related case study on Chisel [7].

II. RELATED WORK

There exist many approaches to bring features from other high level programming languages into hardware design to increase productivity. One approach is to employ a higher level language to serve as a macro processing language to generate predefined blocks written in the underlying HDL language. This simplifies hardware design by providing either parameterization for commonly used hardware, or a simpler, easier syntax. Examples for this approach are Genesis2, which uses Perl to generate SystemVerilog blocks [1], JHDL which is based on Java[2] or HML[3] which uses ML functions. The disadvantages of these methods lie in the lack of connection between the higher level language and the underlying HDL. If no macro for a specific piece of hardware is present, the designer can simply not use it. Another approach involves using a language suited for the domain of the target application. Examples include Esterel [4], which has been modeled for reactive programs and DIL[5], which is an intermediate programming language used to target pipelined reconfigurable architectures like PipeRench. Moreover, there are languages like BlueSpec[6] which is essentially a subset of SystemVerilog putting emphasis on avoiding race conditions by automatically generating scheduling and arbitration logic from a set of "rules" which express synthesizable behavior. These languages are usually designed to support a specific design domain. This, however, leads to these approaches performing poorly when used outside the domain they were intended for.

III. ANALYSIS

This paragraph lists some of the most advertised advantages of Chisel and explores their added benefit over VHDL.

A. Datatypes, Bundles, Interfaces

Chisel allows basic datatypes to be aggregated into bundles to ease usage of bundled signals or create new datatypes. Existing bundles can be subclassed to create a hierarchical structure of datatypes, allowing code reuse and easy revisioning. Bundles can be used in interfaces as well, either by specifying direction at instantiation time (`.asInput/ .asOutput`) or in their definition. Bundles in interfaces can be bulk connected, which makes wiring whole interfaces and busses easy. Moreover the direction can be reversed easily by using the built-in "`.flip`" keyword. VHDL provides similar functionality, called records, but they neither support subclassing, nor can the direction of their subelements be as easily defined or changed as in Chisel. This makes bundling wires and interfacing modules easier in Chisel while also allowing for easy code reuse with hierarchical datatypes. Both Chisel and VHDL support operator overloading for these types, but Chisel makes it easier by automatically inferring bit widths at compile time. This is also useful, if abstract functions described in the next section are used.

B. Functions, Abstraction and Polymorphism

1) Abstract functions

Reusing code requires that code to be as generic as possible, ideally beyond the scope of a specific data type. While functions exist in both Chisel and VHDL, Chisel also allows for parameterized functions which can handle all data types with a common superclass. This concept of polymorphism is not present in VHDL, requiring the hardware

designer to write the same function multiple times for each data type to be used.

2) Functional Instantiation

Chisel also supports functional instantiation to quickly connect small blocks within a larger design. Using a constructor for e.g. a Mux2 object, the inputs and outputs can directly be connected to signals in the surrounding block. No comparable functionality exists in VHDL where modules are always defined as entities which have to be wired at the appropriate place in the code.

C. C++ Simulator

While producing synthesizable Verilog, Chisel can also output a fast C++ simulator of the design. Case studies in [8] show a speedup of up to 8x compared to a state of the art simulation technique. This, however, was strongly dependent on the simulated design, and the amount of cycles. Faster speeds were only achieved when simulating millions of cycles or more, with FPGA emulation being fastest if the simulation exceeded billions of target cycles.

D. Memories and Black Boxes

Since hardware languages cater to a wide variety of target hardware, IP cores provided by the foundry or the vendor usually provide a more efficient implementation than the one that tools can generate from the HDL. To embed IP cores, Chisel provides objects called black boxes. Using a black box in Chisel yields an empty module in Verilog, which can then be implemented by using an external IP core. Unfortunately, no simulation can be carried out if black boxes are used, since they compile into an empty C++ object with inputs wired to their outputs. It would be useful, to be able to either specify C++ behavior and latency for the black box, or provide the ability to embed C++ models for the IP cores which vendors provide. A similar approach already exists for parameterized caches and memories, where parameters like depth and read/write delay can be used to specify detailed timing while simultaneously providing the flexibility to either map to behavioral Verilog or an externally provided instance. VHDL uses vendor supplied block generators for memories and most advanced arithmetic functions. While they provide easy parameterization, the code is usually closed source and not transparent to the hardware designer, who has to resort to the documentation of the used block for details.

IV. Case Study

Since the results of the previous section only explore the advantages of Chisel in theory a case study on Chisel [7] was carried out, where the programming language was evaluated based on a real world implementation task. The Heston Model, used by financial mathematicians to evaluate stock options, was implemented in hardware. During this implementation process, due to the early stage of development of Chisel, more challenges presented themselves:

A. Languagy instability and revisions

Being in active development in its early stages, Chisel syntax and functionality is subject to a lot of changes. This leads to inconsistencies when using papers published early and language tutorials of a more recent date. During the case study, Chisel 2.0 was released, resulting in the aforementioned variety of changes to the syntax and parameters. Ultimately, the previously working implementation of the black box ceased to work, giving irresolvable compile errors. The author believes that these minor issues will be resolved as the language enters future iterations and becomes more stable.

B. Missing functionality/arithmetic

The Heston Model requires a square root to be computed, which was not possible with native Chisel syntax. The final implementation used a black box for the square root, which was later replaced with the Cordic IP block by Xilinx. This, however, made the Chisel implementation unsuitable for simulation, since the square root behavior could not be taken over to the C++ simulator. This problem requires more effort by the creators of Chisel, since the language has to be extended to work around these problems, either by implementing the missing functionality in Chisel itself, or by providing a more sophisticated way to embed external solutions. Because of this, comparisons could only be made in terms of code size and implementation size after synthesis.

V. Conclusion

In this paper, Chisel, and its advertised benefits have been compared to VHDL. In the end, the problems with the early stage of Chisel were pointed out with the help of a case study. At this time Chisel is still in early development and subject to significant changes in terms syntax and function parameters. While it provides advantages in terms of productivity and code size (after an initial education period), the immaturity of the language makes it not yet suitable for use in the industry. Given more time, the author believes that the language is still very promising and suggests a new evaluation once stable syntax and documentation is achieved, because, in its current state, Chisel already provides productivity benefits both in terms of code reuse and general code size reduction with its efficient syntax.

References

[1] Shacham, O. et al. “Rethinking digital design: Why design must change.”

[2] Bellows, P., Hutchings, B. “JHDL – an HDL for reconfigurable systems”

[3] Li, Y., Leeser, M. “HML – a novel hardware description language and its translation to VHDL”

[4] Berry, G., Gonthier G. “The Esterel synchronous programming language: Design, Semantics, implementation”

[5] Budiu, M., Goldstein, S. „Fast compilation for pipelined reconfigurable fabrics“

[6] Bluespec Inc. “Bluespec™ SystemVerilog Reference Guide: Description of the Bluespec SystemVerilog Language and Libraries”

[7] Stumm C., “Investigate the hardware decription language Chisel”

[8] Bachrach J. et al. “Chisel: Constructing Hardware in a Scala Embedded Language”