White Paper - Investigate the hardware description language Chisel

A case study implementing the Heston model

Christopher Stumm, Christian Brugger, Norbert Wehn
Microelectronics Research Group, University of Kaiserslautern, Kaiserslautern, Germany
stumm@rhrk.uni-kl.de, brugger@eit.uni-kl.de, wehn@eit.uni-kl.de

Abstract— This paper presents a case study comparing the hardware description language „Constructing Hardware in a Scala Embedded Language“(Chisel) to VHDL. For a thorough comparison the Heston Model was implemented, a stochastic model used in financial mathematics to calculate option prices. Metrics like hardware utilization and maximum clock rate were extracted from both resulting designs and compared to each other. The results showed a 30% reduction in code size compared to VHDL, while the resulting circuits had about the same hardware utilization. Using Chisel however proofed to be difficult because of a few features that were not available for this case study.

Keywords—Chisel, VHDL, FPGA, Heston Model, Financial Mathematics

I. INTRODUCTION

Hardware description languages like VHDL and Verilog were initially designed for simulation, not synthesis. Therefore these languages contain many constructs that are not synthesizable in modern tools. Another problem is the lack of common techniques that were developed for high-level programming languages during the past decades, like parameterized types, type inference and object orientation.

A. Chisel

In [1], a new hardware description language called „Constructing Hardware in a Scala Embedded Language“(Chisel) was presented. Chisel, being written in Scala, utilizes high-level programming techniques like type inference and object orientation to either create synthesizable Verilog code or a cycle accurate C++-based software simulator. All datatypes and constructs are compiled into an open source library and made publicly available at [2]. For this work Chisel 2.0 was tested. The authors showed a seven to eight times faster simulation time compared to state of the art simulation techniques, as well as only a third of the code size of comparable VHDL designs. In order to use Chisel, one needs only to install the “simple build tool” (SBT), the basic build tool used for Scala projects. SBT mainly features a native compiler for Scala code and manages dependencies between libraries. A further look into the different features of Chisel and a comparison with the feature set of VHDL can be found in [3].

In order to evaluate Chisel’s usability for a real world example the Heston model was implemented in both Chisel and VHDL.

B. Heston model

The Heston Model was published in [4]. It is a stochastic model widely used in financial mathematics to calculate option prices. It describes the changes in an asset’s volatility through a non-deterministic process. Non-determinism was in this case achieved by implementing the model as a Monte-Carlo simulation. During such a simulation several possible ways the stock price could develop, or paths, are calculated in parallel using pseudo-random number generators and a starting stock price. During each time step one new value for the stock price and the volatility are calculated corresponding to the data flow graph (DFG) shown in figure 1. Repeating the DFG for a given number of iterations (100-10000) yields one such path. The implementation in this paper, as described in the following chapter, calculates 15 paths in a pipelined fashion.

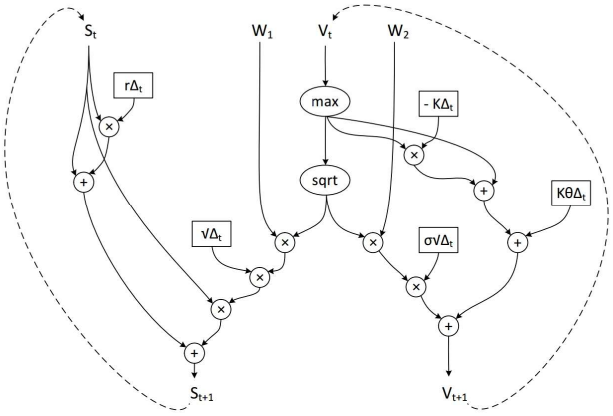

Fig. 1. Data Flow Graph (DFG) of Heston model

II. IMPLEMENTATION

A. Tools

The implemented DFG for the Heston model was emulated on a XILINX Virtex6 FPGA. Therefore the Xilinx ISE Webpack was used for the VHDL designflow and the Verilog code generated by Chisel. As for the designflow in Chisel, a simple

text editor and the aforementioned Scala build tool SBT were employed.

B. Design

The DFG was implemented as a pipeline with fourteen stages. Figure 2 shows an overview for said pipeline.

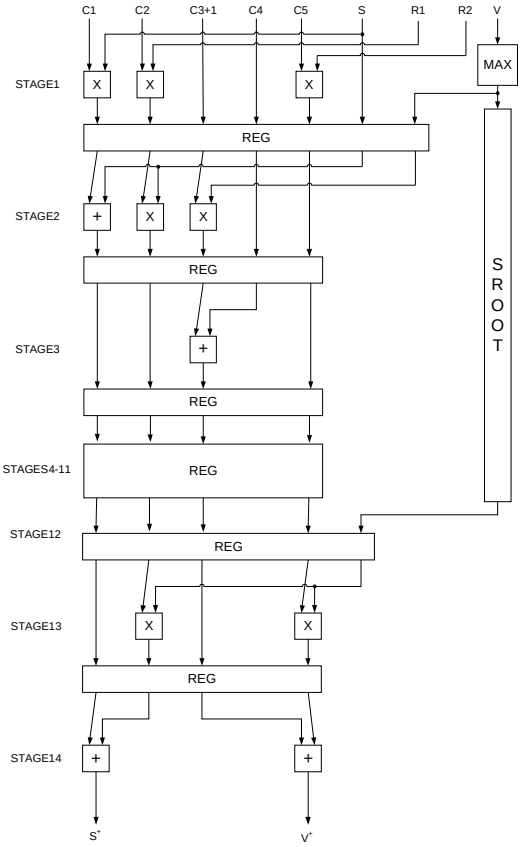

Fig. 2. Pipeline Overview

In order to calculate 15 different paths concurrently a controller based on a finite state machine (FSM) was also designed (not shown in figure 2). The FSM was implemented in different ways for both languages. The VHDL FSM was implemented in one large process. The controller written in Chisel drives most of its outputs through large multiplexers outside the FSM itself. The latter approach requires roughly the same amount of logic as the former, except for the number of registers/flipflops. Figure 3 shows the implementation results for both languages, including the lines of code for the entire design.

	Chisel	**VHDL**
Clock / MHz	107	112
Flipflops	1199	1630
Lookup-Tables	1687	1646
Slices	586	621
DSP-Slices	28	28
Lines of Code	462	667

Fig. 3. Implementation Results

C. Evaluation

One can see in figure 3 that in terms of FPGA utilization the only significant difference between Chisel and VHDL is the number of used flipflops as described above. Considering the codesize in terms of number of lines, one can see that the Chisel code is about 70% the size of the VHDL implementation. While this is nowhere near the 33% mentioned in [1], one has to keep in mind that the design was small to begin with, meaning that this claim might still be true for larger circuits, where more code can be reused.

It is also worth mentioning that, while the Chisel code itself is smaller than the VHDL code, the generated Verilog code becomes very large. In this case study the resulting Verilog code had 1425 lines of code, about three times the amount of the Chisel code. This can become problematic when the designer has to modify the Verilog code as it was necessary in this work because of the following problem: For the "SROOT" unit in figure 2 the CORDIC IP-Core from the XILINX Core Generator was used for the VHDL design. Chisel can't use an external core like CORDIC, and does not yet feature a square root function. Therefore that module had to be replaced by a blackbox in the Chisel code. In Verilog, the code was changed to use the CORDIC unit. Generating a C++ simulator also becomes unavailable when a blackbox is employed in Chisel, thus preventing one of Chisels core features from being used.

Another design choice influenced by a feature that Chisel does not yet support was the datatype used throughout the design. While the XILINX Core Generator can generate circuits for floating point addition, multiplication and square root, Chisel only features signed and unsigned integer types. As a result, one can only implement fixed-point arithmetic (and has to do so by hand as opposed to the automatic generation by the Core Generator), leading to a longer design time.

However, Chisel is still in its early stage of development, with features like support for floating point arithmetics and more complicated math, among others already being in production.

III. Conclusion

Several problems were encountered while implementing the Heston model in Chisel. Some of those can be explained by the early stage of development, like the lack of support for floating point numbers. Others were due to the use of XILINX IP cores in the VHDL design that had to be replaced by blackboxes in Chisel, rendering the very promising C++ unusable. In its current state of version 2.0 Chisel is not very well suited to implementing circuits for arithmetic-intensive models like the Heston model. But given a few more years of development, this language could be a very promising alternative for a decades old language like VHDL.

IV. References

[1] Jonathan Bachrach et al, "Chisel: Constructing Hardware in a Scala Embedded Language", DAC 2012, June 3-7, 2012, San Francisco, California, USA

[2] https://github.com/ucb-bar/chisel

[3] Heilmann F., "Investigate the high-level HDL Chisel"

[4] "A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options", by Steven L. Heston, The Review of Financial Studies 1993 Volume 6, number 2, pp. 327–3