From Chisel to Chips with Open-Source Tools

Martin Schoeberl and Luca Pezzarossa
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Kgs. Lyngby, Denmark
Email: [masca,lpez]@dtu.dk

***Abstract*—Chisel and Verilator provide an open-source stack for digital design. For ASIC synthesis, we have open-source tools like OpenROAD, Yosys, and Magic. OpenROAD is a project to deliver an end-to-end silicon compiler in open source. The aim is to "democratize hardware design" by providing an automated layout generation flow from a design in RTL to GDS files used to produce silicon. Google and Efabless offer free production of chips in a multi-project wafer if the project is available in open source. In this paper, we report our experience in using the open-source tool flow from Chisel to chips with two designs: a small processor using only on-chip resources and a RISC-style processor, called Patmos, with extra external main memory. We have successfully taped out the Patmos processor for the multi-project waver MPW7 from Efabless.**

***Index Terms*—open-source hardware, open-source digital design tools.**

I. Introduction

Open-source software has fueled the development of all aspects of applications and tools. Compilers for all mainstream languages are available in open source. Linux, the leading open-source operating system, powers servers, personal computers, mobile phones, and embedded systems. Web applications' front-end and back-end can be built with open-source tools and libraries. We can browse the web with an open-source browser.

However, in the hardware domain, the situation is very different. Proprietary tools still dominate the design and testing of digital systems. Although we have free tools for FPGA development, we cannot "tinker" with those tools or enhance them by building on top of them. However, for academic research, we would like to have open-source designs [9] and tools available to build upon.

Closed-source compilers also contain the risk that they can introduce spyware into the compiled code. We can also consider synthesis tools as compilers, which compile from VDHL or Verilog to a netlist. It is also imaginable that a malicious synthesis tool introduces a backdoor, e.g., into a chip for a network switch. That issue of a possible backdoor might also be one reason why DARPA initiated the "Intelligent Design of Electronic Assets" program. Furthermore, the program shall enable a no-human-in-the-loop translation from a design described in source code to a final physical layout.

In this paper, we explore the use of the open-source tool flow for chip design. Furthermore, we aim to tapeout two designs: (1) the simple processor Leros [11] using only on-chip memory and (2) the more complex processor Patmos [12] with external memory. The second design was carried out with a group of students during a special course at the Technical University of Denmark.

Our background is in digital design and using FPGAs to verify those designs. We have no background in backend development of ASICs and were hoping that the tool flow has been automated enough so we can handle the full design flow. This paper reports on the experiences during those two approaches.

II. Open-Source Tools

For our projects, we are using open-source tools only, as the title states: from the design in the Chisel language down to GDS files for production. This section gives an overview of the tools we used.

A. Chisel

Chisel is a modern hardware construction language [2], [8]. It is embedded in the general-purpose programming language Scala. Chisel is a domain-specific language, where the domain is the description of digital circuits. We describe hardware in Chisel at the register transfer level. However, the real power of Chisel comes from being embedded in Scala. We can express flexible hardware generators with Scala as an object-oriented and functional language. Functionality expressed in scripting languages such as Perl, TCL, and Python to generate a customizable design in VHDL or Verilog is now possible in the very same language used to describe the hardware.

Chisel is embedded into Scala and executes on the Java virtual machine (JVM). Therefore, it has excellent interoperability with the large pool of existing Scala and Java libraries for design and verification. We can co-simulate a Chisel design with a reference model written in Scala or Java, e.g., using the ChiselVerify project for verification [10], [4]. Furthermore, many modern IDEs, such as IntelliJ IDEA or Visual Studio Code, support Scala and, therefore, Chisel. Autocompletion, exploring the source of the Chisel library, and therefore having the ScalaDoc documentation available is a convenience that software developers are used to but new for classic hardware developers.

Furthermore, we can piggyback on the available infrastructure to distribute open-source libraries. The Maven packet system[1] is a popular software and package management tool. Maven Central[2] is one of the largest repository hosting software

[1]https://maven.apache.org/

[2]https://mvnrepository.com/repos/central

libraries. We can publish hardware components on Maven like any other open-source Java library. Publishing on Maven means that a third-party component can be integrated into the compile flow with a single reference in the `build.sbt` configuration. We consider that the distribution model of intellectual properties (IP)s as JVM libraries from a Maven server has the potential for a small revolution in design reuse in digital design. No more need to manually copy sources from external servers with the danger that the sources get out of synchronization.

Running tests on the Java virtual machine allows using reference models of the hardware design written in Java, Scala, and even in C (via the Java native interface) for co-simulation. For example, in the Leros project, we implemented a reference model as an instruction set simulator in Scala. We execute all self-testing programs on the hardware implementation and the software simulation. In the future, we plan to perform a co-simulation of Leros hardware and software implementation.

B. Synthesis

We may want to add features to existing synthesis tools to develop new synthesis algorithms or write algorithms to generate coarse-grain cell libraries. However, no such synthesis tool existed in open-source. Therefore, Clifford Wolf developed the Yosys Open SYnthesis Suite (Yosys) as part of his Bachelor thesis at the Vienna University of Technology [16]. The focus of Yosys is on high-level digital synthesis. Yosys uses the open-source logic synthesis tool ABC for gate-level optimizations.

ABC [3] is an open-source tool for logic synthesis and formal verification of synchronous digital circuits. ABC supports as input only structural Verilog, which means only the following language constructs are allowed: constant values, wire and port declaration, static assignments, and cell instantiation. We can use Yosys to convert full-featured synthesizable Verilog to structural Verilog. Magic [7] is integrated with ABC. It supports multiple clock domains and more complex components like memories and DSP modules.

C. Simulation

Chisel comes with its own simulator, called Treadle, which can simulate a design in the intermediate representation of FIRRTL. The Chisel simulation is driven by ChiselTest [6], an extension of the Scala test framework ScalaTest. For Verilog, two open-source simulators are available: Verilator and Icarus Verilog.

Verilator [13], [14] is a Verilog simulator that compiles Verilog code to C++ and then to a native binary. Verilator supports synchronous designs only. Therefore, it can deliver faster simulation than event-driven simulators. Verilator claims to have similar or higher performance than the "Big 3" simulators on a single thread. Icarus Verilog [15] is a simulator for Verilog. It compiles circuits described in Verilog (IEEE-1364) into an intermediate format that can then be executed. Icarus Verilog is an event-driven simulator.

We use all three simulators in our design flow. For smaller unit tests and testing the smaller processor Leros, we use Chisel test. For a full system test of Patmos, we use Verilator, with a C-based emulation of external devices, such as external main memory. For the final tests within Caravel, we use Icarus Verilog.

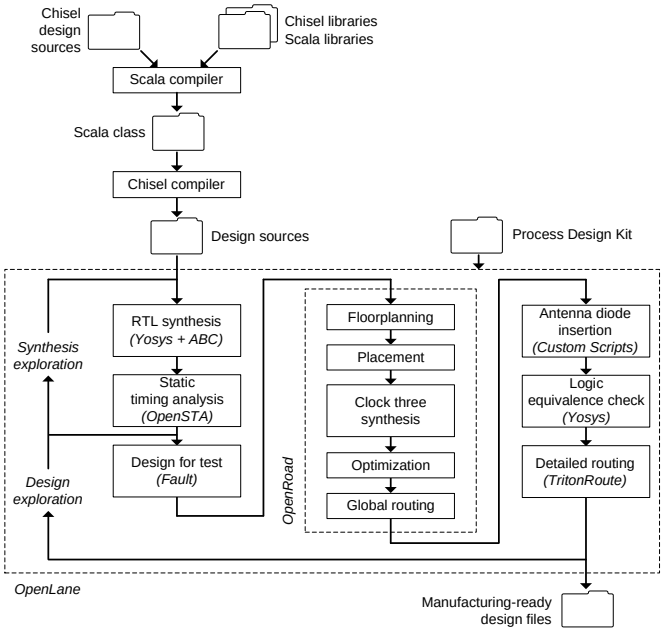

Fig. 1. An overview of OpenLane together with the Chisel tool flow.

D. Open-Source Tool Flows

Figure 1 shows the complete OpenLane [5] toolchain used to synthesize behavioral hardware design using the foundry Process Design Kit (PDK) resources. It includes OpenROAD [1] which is an end-to-end open-source silicon compiler providing automated layout generation flow from a design in RTL (Verilog) to GDS files used to produce silicon.

E. Production Services

Google, together with Efabless,[3] offers free production of chips in a multi-project wafer if the project is available in open-source. In each multi-project waver run, 42 designs are combined on one wafer. If the design is open-source, the production is free of charge and includes five test PCBs with mounted chips. Currently, 130 nm is the only available process. However, GlobalFoundries recently joined Google's open-source silicon initiative with a 180 nm process.

III. The Caravel Design Flow

Google supports the open-source chip design with the open-source PDK for the SkyWater 130 nm fab. The whole process is running through Efabless. Efabless provides the Caravel harness with a user area of about 10 mm^2. The harness contains a RISC-V management core connected to a Wishbone interface to communicate with the user design. The RISC-V core uses an external Flash memory for instructions and a serial interface (UART) to communicate with the external world. The Caravel

[3]http://efabless.com/

project also includes a scripted tool flow using open-source tools and the open-source PDK.

The Caravel project also contains a simple test design consisting of a counter to explore the tool flow. Synthesizing and hardening the harness and the test design are fully scripted and need no manual intervention. Furthermore, Caravel contains pre-checks that must be completed before a tapeout. We managed to harden the test design, perform the precheck, register it at Efabeless, run the prechecks on the Efabless servers, and run the tapeout on the Efabless server. Therefore, making this test design ready for chip production. We managed to run this full flow within just a single day. Of course, we did submit the counter for production. However, we used this test design as the starting point for our designs. Then we could add small parts of the design incrementally and run the flow till tapeout.

IV. The Designs

We developed two independent chips of different complexity: the Leros processor and the Patmos processor. The Leros processor is a simple state machine with datapath. The datapath contains a program counter, an accumulator, an on-chip memory, and an ALU. Leros is intended as a simple, educational processor to serve as a medium complex example in the last chapter of the textbook on digital design in Chisel [8]. Leros is programmed like on bare metal: no operating system or standard library is available. Leros is intended as a small support processor. Therefore, the memories are small enough to fit on the chip. Leros uses dedicated instruction and data memories, a so-called Harvard architecture. The read port of the instruction memory is connected to the instruction fetch of Leros, where the write port is connected to the RISC-V management processor. The RISC-V processor can download a program for the Leros processor and then release the reset of Leros. Leros has as IO devices a serial port and several general purpose IO pins.

Patmos is a 5 stage pipeline, RISC style architecture. Patmos itself is designed to be time-predictable to be used in real-time systems. To support non-trivial programs, the compiler includes a support library (e.g., for integer division and floating point support). Furthermore, we have ported the C standard library to Patmos. The combination of those two libraries consumes about 1/4 MiB of memory, which is too large to be placed on-chip. Therefore, we need external memory. Patmos has instruction and data caches. Both caches are connected to a memory controller and external shared memory. However, the Caravel chip has only 32 IO pins, which are too few to connect a parallel memory. We, therefore, use an external DRAM with an SPI interface.

If both designs fail, e.g., we made a mistake with the memories, we have a small backup design to see a chip work. We added a small state machine to the design. That state machine writes "Hello World!" in Morse code on one pin. So the absolute minimum is an LED that sends text in Morse code.

V. A Class Project

In the spring semester of 2022, we organized a special course to explore open-source design tools and build a Patmos chip. Twelve students signed up for this course. Most of the them were Electrical Engineering Bachelor students in their 4th semester. They have as background two semesters of digital design in VHDL and Chisel.

The task has been to integrate Patmos into the given Caravel framework. As the usable pin count is just 32 pins, we needed to find a solution for external memory. We selected a DRAM with an SPI interface. The students developed an SPI interface in Chisel and a memory controller that translates between the Patmos burst interface and the SPI commands. They tested Patmos with this external memory in an FPGA. We finalized the whole process within one semester and recently finished the tapeout.

In the Patmos project with the student group, we have been not careful enough to build up the design incrementally. As twelve students worked in parallel, we had a classic integration phase at the end. Only after the integration, we run hardening. The prechecks reported several errors. The final phase of fixing all those errors took way longer than expected—we were working until the deadline of the MPW-6 run and missed it by a few hours. Therefore, we submitted the design for the following MPW-7 run.

Overall, the class project was a success. Seeing the option to get a real chip back from a class project was highly motivating for the students. They worked hard to solve all those challenges and overcome frustrations during development. The project is hosted on several repositories on the GitHub organization at https://github.com/os-chip-design. The course has received press coverage.[4]

VI. Lessons Learned

We started the exploration of open-source tool flow without prior knowledge of the ASIC *backend* flow. We are mainly used to test our designs in FPGAs, wherein 90 % of the cases synthesis, place and route, and bitstream generation is pressing a single play button. DARPA's research project "Intelligent Design of Electronic Assets" claims a similar experience for ASIC designs:

> To overcome the design expertise gap and keep pace with the exponential increase in chip complexity, the Intelligent Design of Electronic Assets (IDEA) program seeks to develop a general-purpose hardware compiler for no-human-in-the-loop translation of source code or schematic to physical layout (GDSII) for SoCs, System-In-Packages (SIPs), and Printed Circuit Boards (PCBs) in less than 24 hours.

The initial issues have been setting up the toolchain and installing the correct versions of various tools. Since some of those open-source projects are moving quite fast, their dependencies also change. However, the documentation lacks

[4] https://www.compute.dtu.dk/english/news/nyhed?id=ed010284-154f-43a3-a352-6dce97eaf72c

behind. We had to ask in busy Slack channels and got friendly, detailed help.

For tiny designs, like the counter example from Caravel, the flow already works surprisingly well. We can generate GDS files from the Verilog code in less than an hour. We can perform precheck and tapeout in less than an afternoon. However, with the medium complex design of a processor with on-chip memories, there was some manual manipulation necessary. In this process, we needed to dig deep into the configuration files and the error logs.

Furthermore, we were surprised by the time and memory necessary to synthesize medium-sized designs. The Patmos processor synthesizes for an FPGA in the range of minutes, but for Verilog to GDL, it was more in the range of several hours. In future work, we need to learn how to partition a design for hardening individual components. This might be trickier with a Chisel-based design, as it spills out a single Verilog file.

In the greater Copenhagen area, we have several chip design companies and high demand for chip design and verification engineers. In the last years, we did not deliver enough engineers and the companies have been forced to hire engineers from Europe and further away. Using open-source tools and a free chip service from Google enables a new dimension in teaching ASIC design. With our course, we saw very motivated students probably specializing in chip design and becoming future engineers for Denmark's booming chip design industry.

VII. Conclusion

Recent advances in open-source synthesis tools enable a full open-source flow for digital designs from the description in a hardware language down to the GDS files for chip production. Google offers free chips on the 130 nm SkyWater fab for an open-source project. We have explored that flow with two different designs: the tiny sequential processor Leros and the time-predictable RISC processor Patmos. The RISC processor and the adaption for the Caravel project was a one-semester class project at the Technical University of Denmark. We managed to successfully tapeout the Patmos processor.

Source Access

Both projects are open-source and available on GitHub: https://github.com/leros-dev/leros-chip and https://github.com/os-chip-design/patmos-chip are the Caravel projects. The Chisel source of the projects are at: https://github.com/leros-dev/leros and https://github.com/t-crest/patmos.

References

[1] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, Geraldo Pradipta, Sherief Reda, Mehdi Saligane, Sachin S. Sapatnekar, Carl Sechen, Mohamed Shalan, William Swartz, Lutong Wang, Zhehong Wang, Mingyu Woo, and Bangqi Xu. Toward an open-source digital flow: First learnings from the openroad project. In *DAC*, page 76. ACM, 2019.

[2] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avizienis, John Wawrzynek, and Krste Asanovic. Chisel: constructing hardware in a scala embedded language. In *The 49th Annual Design Automation Conference (DAC 2012)*, pages 1216–1225, San Francisco, CA, USA, June 2012. ACM.

[3] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 24–40, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[4] Andrew Dobis, Tjark Petersen, Hans Jakob Damsgaard, Kasper Juul Hesse Rasmussen, Enrico Tolotto, Simon Thye Andersen, Richard Lin, and Martin Schoeberl. Chiselverify: An open-source hardware verification library for chisel and scala. In *2021 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, 2021.

[5] Ahmed Ghazy and Mohamed Shalan. Openlane: The open-source digital asic implementation flow. In *Workshop on Open-Source EDA Technology (WOSET)*, 2020.

[6] Richard Lin. ChiselTest. https://github.com/ucb-bar/chisel-testers2.

[7] Alan Mishchenko, Niklas Een, Robert Brayton, Stephen Jang, Maciej Ciesielski, and Thomas Daniel. Magic: An industrial-strength logic optimization, technology mapping, and formal verification tool. In *Proc. IWLS'10*, 2010.

[8] Martin Schoeberl. *Digital Design with Chisel*. Kindle Direct Publishing, 2019. available at https://github.com/schoeberl/chisel-book.

[9] Martin Schoeberl. Open-source research on time-predictable computer architecture. In *Proceedings of the 25th Euromicro Conference on Digital System Design (DSD)*, 2022.

[10] Martin Schoeberl, Simon Thye Andersen, Kasper Juul Hesse Rasmussen, and Richard Lin. Towards an open-source verification method with chisel and scala. In *Proceedings of the Third Workshop on Open-Source EDA Technology (WOSET)*, 2020.

[11] Martin Schoeberl and Morten Borup Petersen. Leros: The return of the accumulator machine. In Martin Schoeberl, Thilo Pionteck, Sascha Uhrig, Jürgen Brehm, and Christian Hochberger, editors, *Architecture of Computing Systems - ARCS 2019 - 32nd International Conference, Proceedings*, pages 115–127. Springer, May 2019.

[12] Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: A time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, April 2018.

[13] Veripool. Verilator. https://www.veripool.org/wiki/verilator.

[14] Veripool. Verilator manual. https://www.veripool.org/wiki/verilator/Manual-verilator, 2020.

[15] Stephen Williams. Icarus Verilog. http://iverilog.icarus.com/.

[16] Clifford Wolf. Design and implementation of the yosys open synthesis suite. Bachelor thesis, Vienna University of Technology, 2013.