Free/Open Source EDA Tools Application in Digital IC Design Curricula

Aleksandar Pajkanovic, *Member*, *IEEE*

***Abstract*—This paper represents a report on how free and open source software EDA tools may be used to organize a digital integrated circuits design course at the university level, without any financial investments in licenses. The course is built around several publicly available processor cores. These are of different complexity so first intrinsic properties are investigated. Then, using more complicated designs we examine how to increase performance through pipeline and cache associativity configurations. In this way we introduce RISC-V ISA and Chisel into the curricula. Finally, we provide a short overview of tools for automated design, from RTL all the way to silicon.**

***Index Terms*—open source, digital design, RISC-V, Chisel**

I. Introduction

Since the first implementation of a transistor was demonstrated at the Bell Labs back in mid-20th century [1], the industry has shifted from traditional, as established by the Industrial Revolution, to an economy based on the information technology (IT). That event, we understand today, represents the onset of the Information Age–the age characterized by rapid growth and development in all areas of life, driven by the semiconductor industry. Its workhorse, the CMOS technology process, is characterized by low power consumption, extreme scalability and ease of mass production. The ability to implement an idea, a solution using this technology, i.e. the ability to design integrated circuits (IC), or chips, has been of greatest importance for decades and it will be even more so in decades to come; namely, even though the Moore's Law [2] has ended–i.e. we do not advance our ICs by scaling anymore, but rather using advanced techniques [3] in design and verification phases, in order to achieve better performance–we still fabricate our solutions in silicon CMOS technology process.

Over the last five years, we have been establishing IC courses at the Faculty of Electrical Engineering in Banja Luka. This effort has been reported in [4], [5] and has recently culminated in fabrication ready circuits [6]. All this has been achieved using exclusively free and open source EDA software tools, with the help of many contributors - both students and tool developers. In those developments, however, main effort was in the analog domain, whereas in this particular paper we focus on digital IC design course, the examples it's built upon and tools used.

Since materials such as combinatorial and sequential circuits are covered in other courses, for this particular course we've decided to learn about more advanced concepts using the free intellectual property (IP) available in the community. Thus we study three processor designs, we learn a new approach to digital design (hardware construction, instead of description) and we drive the RTL code all the way to GDS, i.e. silicon ready file, using the free and open source toolchain.

In the next section we present motivation for writing up this paper, then we provide brief overviews of the RISC-V Instruction Set Architecture (ISA), *Chisel* - the hardware construction language (HCL) and processor cores we learn about and use to demonstrate theoretical concepts. Finally, in the sixth section, we present the free/open source digital IC design toolchain.

Aleksandar Pajkanovic is with the Faculty of Electrical Engineering, University of Banja Luka, Patre 5, 78 000 Banja Luka, The Serbian Republic, Bosnia and Herzegovina, e-mail: aleksandar.pajkanovic@etf.unibl.org

II. Motivation

There is no point in living during the Information Age, unless we are going to use its benefits. While boundaries are important parts of our lives and some should never be crossed, there are those boundaries that are not actually natural - these simply existed due to the fact that we knew not how to overcome them. This is not the matter of destroying those boundaries, but rather outgrowing them to improve the world and general quality of life. Such borders are those related to knowledge. With the Internet and its omnipresence - knowledge is omnipresent as well, for those who seek it. Our idea is to build the IC curricula by standing on the shoulders of the giants–of those who have had the chance to grow and develop for decades in this domain, thanks to another kind of boundaries. Therefore, we bring what's best on this planet right in our own court and thus enable our own students right here in Banja Luka to gain world-class expertise in the semiconductor industry, which, after all, is the most sophisticated commercially available technology process. And we do that without any financial investments - but rather simply: by reaching out.

Main motivation behind this paper is to contribute to the open source hardware community by sharing collected experiences and provide feedback on a subset of freely available tools and IPs, for all those who find themselves struggling to get started, at no cost, in this extremely interesting and exciting engineering and science area.

III. RISC-V

Computer architecture is, as most engineering areas nowadays are, an incredibly vast discipline. However, for the purposes of this short article, we will overly simplify and point out that it can be divided in two subdiciplines: software and

hardware. ISA is probably the most important interface in the universe as it serves to connect these two worlds. This has become clear with the IBM 360 appearance on the market, when the concept of ISA was first introduced. All ISAs with significant usage are proprietary, which does make sense when taking into account that the creators protect their intelectual property. However, *open* standards (the correct term would be open-source - to maintain analogy with software, where it all started, but there's no source code nor source files in this context, so we just call them open standards) like ethernet have proven successful. Successful, meaning that we have seen free-market competition through technical improvements, whence the end-users benefited the most [7]. Therefore, it is a crucial question to raise: why not create an open standard for the most important abstraction layer, the ISA?

In an answer, RISC-V (where V is a roman number five, thus pronounced) ISA has been designed - a completely free and open ISA, built and improved on the original RISC architectures. While it was designed originally for research and teaching, it is on its road to become a standard for industry implementations, as well. The final result is a simple and modular ISA, well suited to both high-performance systems and low-power embedded controllers. This is enabled by dividing the instructions into the obligatory base subset of ISA, present in any implementation, plus the optional extensions (subsets). The base is restricted to contain a minimum number of instructions sufficient for compilers, assemblers and linkers. If an operating system is to be used, an additional subset is required. Therefore, it is possible to look at the RISC-V as if it is actually a family of related ISAs. There are two base integer variants, RV32I and RV64I, each providing 32-bit or 64-bit width, respectively. The subset marked with E is designed for small microcontrollers, whereas the subset M enables multiplication and division, F is single-precision floating-pint, D for double-precision, etc. The most available RISC-V processors implement the RV64IMAFD flavor, which, due to its popularity, is marked with G (for general) [8]. Such is the rocket core, discussed in Section V of this paper.

RISC-V represents a perfect combination of simplicity and industrial application, and is, therefore, selected as the ISA to be studied in VLSI courses.

IV. Chisel

Chisel is a hardware construction language, first introduced in [9], built as a domain specific language (DSL) embedded in Scala, with the idea to support advanced hardware design by providing important concepts established in software engineering. These include object orientation, functional programming, parameterized types and type inference. Such powerful abstraction features enable high level of code reuse, thus improving efficiency in constructing new hardware systems. In this way, while not claiming that Chisel is *better* in general, we do point out that Chisel does provide new paradigms in hardware design, that can increase productivity.

At first glance, looking at simple combinatorial module such as a multiplexer shown in Listing 1, there are no conceptual differences to standard Verilog HDL approach, aside from the syntax [10]:

```
class Mux2 extends Module {
  val io = IO(new Bundle {
    val sel = Input(UInt(1.W))
    val in0 = Input(UInt(1.W))
    val in1 = Input(UInt(1.W))
    val out = Input(UInt(1.W))
  })
  io.out := (io.sel & io.in1) |
              (~io.sel & io.in0)
}
```

Listing 1. Multiplexer 2/1 in Chisel

While some strangeness is present, due to inheritance and `:=` operator, it is quite obvious we have a group of input/output ports packed in a `struct`-like piece of code (a *Bundle*) and some wiring in the last line. Looking at state elements, Chisel is still quite comparable to Verilog, as demonstrated by the 4-bit shift register in Listing 2 [10]:

```
class ShiftRegister extends Module {
  val io = IO(new Bundle {
    val in  = Input(UInt(1.W))
    val out = Output(UInt(1.W))
  })
  val r0 = RegNext(io.in)
  val r1 = RegNext(r0)
  val r2 = RegNext(r1)
  val r3 = RegNext(r2)
  io.out := r3
}
```

Listing 2. 4-bit Shift-register in Chisel

Of course, there are plenty of details to unpack here, such as the casting to `W` (bit width) type, wiring implicated from using `RegNext` specifically and so forth, but these are beyond the scope of this paper. For further instructions on Chisel, the reader is referred to [10], [11]. In [10], Chisel and dependencies are installed to a local machine and a set of examples, problems and solutions is provided, while the basics are explained through an online, github-wiki-based tutorial. In [11] Chisel is presented through an interactive tutorial based on a jupyter notebook with assignments and instructions intervened. There's an online version of the bootcamp, as well.

Chisel, rather, its toolchain, is capable of emitting Verilog for three different backends to generate either: (a) a software simulator that can be fed binaries, (b) a bitstream for an FPGA, or (c) netlist to be further used by automatized toolchain to yield GDS mask patterns ready for ASIC production. The actual advantages of Chisel over today's HDLs are shown in the next section, where processor cores used in our course are presented.

V. Processor Cores

Students get to learn about computer architecture and VLSI design by building one simple core, then applying that knowledge and expanding on it by investigating and modifying two more cores, each of these with several flavors.

A. Hack

Hack is the name of a computer and that computers processor - both developed within the now famous *Nand2Tetris* course, taught by Shimon Shocken and Noam Nisan. The course covers all abstraction levels starting from designing basic combinatorial modules, starting from NOT and XOR gates, collecting them into subsystems such as memories and aritmetic logic units (ALU), all the way through writing assembly code, operating system and even designing a complete java-like language, writing its compiler and then using the whole pyramid to design and play games. All tools required for the course are available free of charge at the course webpage [12], as well as the companion book [13], and the course itself, having been taught at several famous universities, is available at coursera.org.

We use the first half of Nand2Tetris, where students through six project assignments develop the chipset first, then the ALU, registers, RAM and program counter and learn about assembly code. Finally, they put all these together to obtain a fully functional Harvard architecture-based computer. In this way, a very sound foundation is created, making each candidate ready for tackling more complex problems such as memory-mapped input/output (MMIO) peripheries, pipeline, multicore processors, SoC design, etc.

B. Sodor

Sodor [14] is a collection of five simple and open-source cores written in Chisel, developed and published to be used at university level courses, as the basis for practical examples based on the theory presented in [3] and taught at UC Berkeley, *CS152/252A* courses [15].

After having developed Hack completely using the HDL provided at [12], and then taking a week to learn Chisel basics via the bootcamp [11], students learn about a real, RISC-V compatible processor. We focus on 1- and 5-stage implementations, as these provide enough to learn about performance metrics, most important benchmarks, measuring and comparing thus yielded numbers and introduce pipeline.

C. Rocket-chip

All the cores available in Sodor collection are written in ”plain” Chisel, meaning that no advanced principles such as core parameterization via diplomatic design patterns [16] are utilized to fully leverage advantages of Chisel over the standard HDLs.

Rocket-chip [17] is an open-source SoC generator also developed at the UC Berkeley, with the difference to Sodor cores that hardware it generates has actually been fabricated in silicon; therefore, while useful for research and teaching, these designs are applicable in industry, as well. As with other Chisel designs, its output is synthesizable Verilog and in this codebase the main advantages of Chisel are demonstrated.

It it important to differentiate between the *rocket* core as one of the cores that may be used within the SoC generated by the *Rocekt-chip* generator, whereas different cores may be used as well. Furthermore, the actual capabilities and performance may be fine tuned through the configuration parameters. This level of flexibility is yielded by the system's modular design, built upon HCL approach to hardware design.

To learn about specifics of the following example in detail, we refer the reader to Chipyard documentation [18], as such discussion is beyond the scope of this paper. Here, we list a few lines of code in order to demonstrate the agility of the SoC generator, Chisel and the approach in general. The default configuration of a Rocket-chip will yield a single 64-bit core accompanied by a floating point unit (FPU) and with level 1 cache. However, if it is required to generate core with smaller cache and without the FPU (lower the energy consumption, decrease area), by inspecting the `Configs.scala` file an appropriate configuration may be found:

```
class DefaultSmallConfig extends
    Config(new WithNSmallCores(1) ++
    new WithCoherentBusTopology ++
    new BaseConfig)
```

Listing 3. Configuration to gneerate singlecore system without FPU and with smaller cache

Next, just by running:

```
make CONFIG=DefaultSmallConfig
```

we obtain synthesizable Verilog and a software simulator, with characteristics defined by the configuration line above. Similarly, configurations with dual (or more) cores may be used, at 32- or 128-bit widths, etc.

We do not delve into diplomatic patterns that enable such level of modularity, but rather follow examples from UC Berkeley [15], where the students in their first computer architecture/VLSI course develop understanding of the mutual dependency between hardware design and its application, i.e. software that it executes. Hence the projects for this part of the course are related to the features of the pipeline and cache: we ask for performance measurement while a specific piece of code is executed, then we seek ways to optimize hardware design by, say, changing cache associativity, and, finally, the benchmarks obtained after the modifications are compared - number of misses and instruction per cycle (IPC), in this particular example.

VI. Automated Toolchain

Once the design is settled from the computer architecture point of view, i.e. either we've reached the requirements or the time is simply up, it is time to look for ways to materialize the idea in a real circuit. While FPGA is a valid destination for Chisel code, this is not the topic of VLSI courses in general, so we focus on the ASIC digital synthesis toolchain within this paper. That is a set of software tools used to transform the Verilog (emitted by Chisel in this case) netlist into a physical digital circuit. In semiconductor industry today, these tools are vast proprietary suites developed and licensed by large companies such as Cadence or Synopsis. These are expensive to the point that quite a limited number of long running IC manufacturers may obtain the licenses on a regular basis. There are university and start-up programs, but those are also

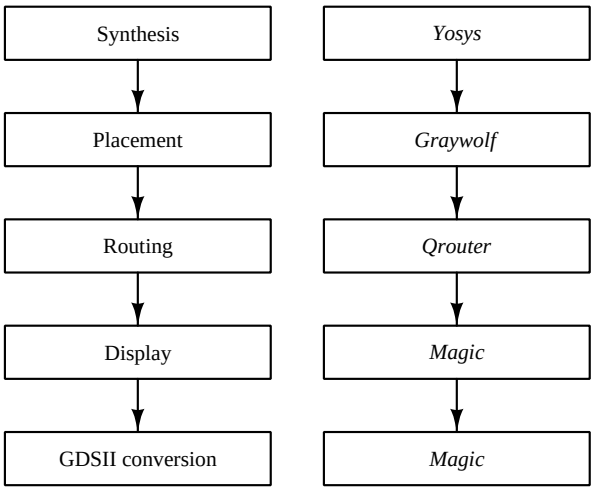

Fig. 1. Digital IC design open source toolchain: a) general approach, and b) tools applied during this course

far from free of charge for small universities or a team of two just starting out.

Qflow [19] is a complete, free of charge and open-source toolchain for synthesizing digital circuits starting from Verilog source and ending in physical layout for a specific target fabrication process. While the process is more detailed, we present a simplification describing only the major steps - rougly shown in Fig. 1. The first step in the automation process is to map the netlist onto a standard cells library, colloquially referred to as *PDK* (stemming from *project design kit*). PDKs are a topic of its own and a complex one, while at that, since these are also proprietary in general. Recently, there have been revolutionary development with SkyWater PDK [20], but not in time to be included in the course edition we are reporting on with this paper. In previous iterations of the course scalable CMOS PDK [21] was used. For now, we keep to the open-source PDK provided by the Oklahoma State University (OSU). This step is done by yosys [22]. Next, the design is to be placed and routed. In shortest terms, this when the standard cells are spread across the available area, while grouped in blocks and interconnected (routed). Graywolf [23] is the member of the qflow toolchain that does the placement, while routing is performed by qrouter [24]. Finally, for layout inspection, DRC and GDS generation Magic [25] is used. Qflow, nor the provided PDK are not capable of creating a microprocessor that may compete with current 3 GHz+ multicore server processors, but these tools will successfully handle simpler designs that may be found in SoC all over the market - such as SPI, for example. Firt live demonstration of a chip fabricated using nothing but qflow is presented in [26].

VII. Conclusion

While ISAs and processor cores are not directly a subset of a VLSI related course, we do live at a revolutionary moment in technology history - free and open source tools and PDKs are a reality, hence ASIC design and fabrication are within reach to individuals, start-ups and small universities with very limited funds. In future iterations of these courses, we plan to improve our automated design flow replacing qflow with openLane and including the open-source RAM compiler, openRAM. To overcome the steep learning curve of the new hardware design paradigm introduced with Chisel and rocket-chip, we are envisioning a Chisel GUI. Finally, we hope to fabricate students' designs through SkyWater 130 nm process, an open-source PDK.

References

[1] W. Shockley, "The theory of p-n junctions in semiconductors and p-n junction transistors," *Bell System Technical Journal. Institute of Electrical and Electronics Engineers (IEEE)*, vol. 28, no. 3, pp. 435–489, 1948.
[2] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, 1965.
[3] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach, 6th*. Morgan Kaufmann Publishers Inc., 2019.
[4] A. Pajkanovic, "On the application of free CAD software to electronic circuit curricula," in *Proc. of 3rd International Conference on Electrical, Electronic and Computing Engineering IcETRAN 2016*, Zlatibor, Serbia, 2016, pp. ELI1.3.1–4.
[5] A. Pajkanovic and Z. Ivanovic, "A report on recent development in application of free CAD software to IC curricula," in *Proc. of 5th IcETRAN 2018*, Palic, Serbia, 2018, pp. 847–851.
[6] A. Pajkanovic, "CMOS IC from schematic level to silicon within IC curricula using free CAD software," in *Proc. of INDEL 2020*, Banja Luka, Bosnia and Herzegovina, 2020.
[7] A. Waterman, "Design of the RISC-V instruction set architecture," Ph.D. dissertation, University of California at Berkeley, 2016. [Online]. Available: https://people.eecs.berkeley.edu/ krste/papers/EECS-2016-1.pdf
[8] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual, vols. i-ii," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-118, May 2016.
[9] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC2012*, San Francisco, USA, 2012.
[10] chisel-tutorial. [Online]. Available: https://github.com/ucb-bar/chisel-tutorial/wiki
[11] chisel-bootcamp. [Online]. Available: https://github.com/freechipsproject/chisel-bootcamp
[12] nand2tetris. [Online]. Available: nand2tetris.org
[13] N. Nisan and S. Shocken, *The Elemenets of Computing Systems*. MIT Press, 2008.
[14] riscv-sodor. [Online]. Available: https://github.com/ucb-bar/riscv-sodor
[15] Computer Architecture 152/252A, spring 2021. [Online]. Available: https://inst.eecs.berkeley.edu/ cs152/sp21/
[16] H. Cook, "Diplomatic design patterns: A TileLink case study," in *CARRV17*.
[17] K. Asanović, R. Avižienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, P. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, , and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, MA, Tech. Rep. Technical Report UCB/EECS-2016-17, 2016.
[18] Chipyard. [Online]. Available: https://chipyard.readthedocs.io/
[19] Qflow. [Online]. Available: http://opencircuitdesign.com/qflow/
[20] SkyWater. [Online]. Available: https://www.skywatertechnology.com/
[21] MOSIS. [Online]. Available: https://www.mosis.com/
[22] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.
[23] "Graywolf," https://github.com/rubund/graywolf.
[24] "Qrouter," http://opencircuitdesign.com/qrouter.
[25] Magic. [Online]. Available: http://opencircuitdesign.com/magic
[26] T. Edwards and M. Kassem, "The Raven Chip: First-time silicon success with qflow and efabless," in *Free Silicon Conference 2019, FSiC 2019*, Paris, France, 2019. [Online]. Available: https://wiki.f-si.org/index.php/FSiC2019