What You Simulate Is What You Synthesize: Design of a RISC-V Core from C++ Specifications

Simon Rokicki, Davide Pala, Joseph Paturel, Olivier Sentieys

What You Simulate Is What You Synthesize: Design of a RISC-V Core from C++ Specifications

Simon Rokicki, Davide Pala, Joseph Paturel, Olivier Sentieys
Univ Rennes, Inria, CNRS, IRISA

***Abstract*—Designing the hardware of a processor core as well as its verification flow from a single high-level specification would provide great advantages in terms of productivity and maintainability. In this work we highlight the gain of starting from a unique synthesis and simulation C++ model to design a RISC-V core. The specification code is used to generate both the hardware target design through High-Level Synthesis as well as a fast and accurate simulator of the latter through software compilation. The object oriented nature of C++ greatly improves the readability and flexibility of the design description compared to classical HDL-based implementations. The processor model can easily be modified, expanded and verified using standard software development methodologies. The main challenge is to deal with C++ based synthesizable specifications of core and uncore components, cache memory hierarchy, and synchronization. In particular, the research question is how to specify such parallel computing pipelines with high-level synthesis technology and to demonstrate that there is a potential high gain in design time without jeopardizing performance and cost. Our experiments demonstrate that the core frequency and area of the generated hardware are comparable to existing implementations.**

I. Introduction

Since decades software and hardware have shared a symbiotic relationship - a situation where both elements support and need each other to thrive and progress. Innovations in hardware architecture are leading to better processors which can support the complex software applications being developed today. Similarly, software and programming languages have made it easier to prototype, simulate and even synthesize hardware, for example the creation of High-Level Synthesis (HLS) tools. HLS is a hardware design technique where an algorithm written in a high-level language like C or C++ is interpreted to create digital hardware which implements the same functionality. HLS tools are very useful for designing complex controllers or accelerators, as they allow engineers to focus simply on functionality, without worrying about architectural details too much.

Although High-Level Synthesis is making huge progress in dealing with complex structures, how far can these tools go? Can they be used to design something as complex as a microprocessor? In particular how to specify such parallel computing pipelines (e.g., core pipeline stages, cache hierarchy, communications with uncore components) with High-Level Synthesis technology and to demonstrate that there is a potential high gain in design time without jeopardizing performance and cost. This work is a first attempt to answer these questions.

II. Comet Design Flow

This paper describes Comet [1], an in-order micro-architecture supporting the 32-bit RISC-V instruction set (rv32i) which has been designed from a single C++ specification using High-Level Synthesis (HLS) tools. Figure 1 summarizes the design flow of Comet. On the ASIC part, Mentor Catapult HLS and a 28 nm CMOS technology library have been used, together with Synopsys Design Compiler and Modelsim for logic synthesis and validation. The design has been validated with a set of RISC-V benchmarks and the core has been synthesized down to the gate-level. The Comet core can also be synthesized for an FPGA target using Vivado HLS. The current version of the core has been synthesized and tested on a Xilinx XC7Z045 FPGA.

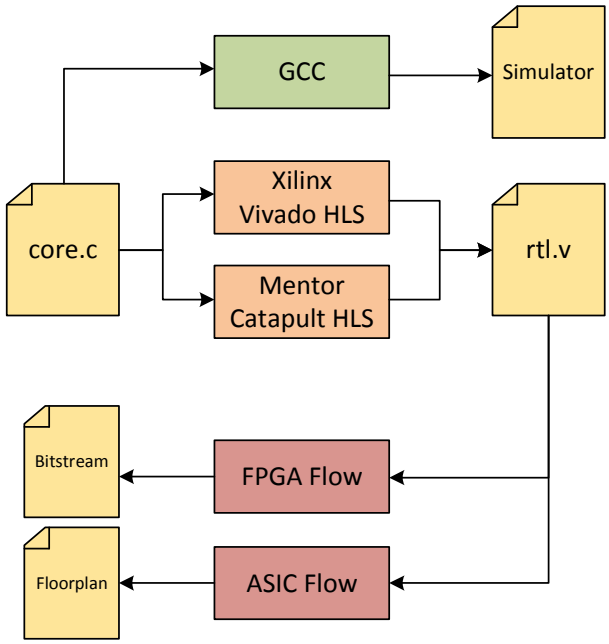

Fig. 1. Description of the design flow used for the Comet core. The flow only uses C++ source files as input and uses HLS tools to generate an RTL description of the core. This HDL file can be mapped into FPGA or ASIC. The same source files can be compiled to generate a fast, cycle and bit-accurate simulator of the core.

High-level synthesis includes scheduling and binding phases, mainly suitable to target parallel datapaths controlled by state machines. To circumvent this primary usage, there is a need to mislead the synthesis flow by explicitly specifying the dependencies in the core pipeline stages. Therefore, the C++ description of the processor is a while loop which calls functions and methods corresponding to the five pipeline stages (Fetch, Decode, Execute, Memory and Write-Back). The Decode and Execute stages are based on *switch/case* statements. Resource sharing between the different *cases* is handled by the HLS tool but can be improved using techniques such as datapath merging [2]. The core also supports forwarding and stall mechanisms. Memory interfaces are defined by

inheritance of an abstract class, exposing a defined set of I/O routines to the core, making the latter agnostic to the memory access mechanisms. This approach allows for easy switching between scratchpads, caches or any combination of them.

Basing the development flow on High-Level Synthesis presents several advantages:

- The C++ description can be compiled to generate a fast, cycle and bit-accurate simulator of the core. Thanks to HLS, we can ensure that this simulator is equivalent to the generated RTL. Moreover, the core can be tested and debugged at the C++ level (using well known tools such as GDB). Debugging at the HDL level is much more labor intensive.
- Designing and modifying the core at the C++ level is much simpler than at the HDL level. The core description is only 1000 lines of C++ code and can be easily understood and modified. Modifying it to print execution traces or to instrument the execution (e.g., counting instruction occurrences) is straightforward.
- Several software development techniques can be used to simplify the development process. For example, our project uses standard Continuous Integration tools to verify, at each modification, that the simulator works properly and can be synthesized into hardware. Performing all these tests on a standard hardware development flow would be much more challenging and time consuming.

III. Simulator Performance and Support

Additional C++ code (which does not aim to be synthesized) is used to open elf files, emulates system calls or instrument the execution. Consequently, a broad range of bare metal and OS supported applications can be executed by the simulator. To measure the simulation performance, several applications from the MiBench benchmark suite have been selected (the set of workloads has been picked from all the categories of the suite). Using a modern computer equipped with an 8th-generation Intel core i7 CPU, an average performance of 26 Millions cycles per second has been recorded. Comparatively, the verilator simulator of the Rocket chip executes approximately 23 thousands cycles per second.

Thanks to these properties, Comet is currently used internally as a test platform for several research projects around Non-Volatile Processors and Fault-Tolerant Multicores.

IV. Synthesis Results

Having the high-level description equivalent to the synthesis hardware model limits issues with the maintenance of an RTL source and provides productivity gains. However, a decent question would arise about the efficiency of the resulting hardware. In this section, we show that our model is competitive in terms of area and operating clock frequency. Core performance is not studied here. As the micro-architecture used in Comet is similar to the one of Rocket, the performance should be equivalent.

The Comet C++ description has been synthesized using Mentor Catapult HLS and Synopsys Design Compiler. The technology targeted is a 28 nm CMOS FDSOI from ST Microelectronics. The core is synthesized targeting 700 MHz, the limiting factor being the low-power memory library used in our setup. For a fair comparison, we also synthesized other cores using the same setup. The results are summarized in Table I. We can see that the area of Comet core is similar to those described using Verilog or Chisel. Note that Rocket is slightly bigger than others because it supports the M extension.

TABLE I
Area and frequency results for different RISC-V cores.

Core	Language	Frequency Target (MHz)	Area (μm^2)
Comet [1]	C++	700	8 476
PicoRV32 [3]	Verilog		7 830
Rocket [4]	Chisel		11 764

V. Conclusion

The work done with Comet highlights the benefits of using HLS to develop CPU cores since it significantly reduces development and debugging time. Moreover, this also provides a cycle and bit-accurate simulator which is, by construction, equivalent to the hardware core. This work demonstrates that HLS tools are mature enough for handling complex, control-dominated code such as a pipelined cores. The resulting hardware presents similar area and operating frequency to HDL written cores. Comet is fully open-source and can be found at https://gitlab.inria.fr/srokicki/Comet.

However, we also observed some limitations with state-of-the-art HLS tools. Indeed, we had to explicitly describe the organization of the pipeline as well as the forwarding logic. We believe that this kind of optimizations could be done automatically by source-to-source transformations. As a future work on this project, we plan to develop more complex micro-architectures (ISA extension, interrupt controller for OS support, out-of-order processor, shared-memory multiprocessor) to see how far the limits of HLS can be pushed.

Acknowledgements

The authors thank Valentin Egloff, Edwin Mascarenhas and Gurav Datta for their technical involvement and their contributions in the Comet project.

References

[1] S. Rokicki, J. Paturel, D. Pala, E. Mascarenhas, V. Egloff, and O. Sentieys, "Comet: A pipelined RISC-V Processor generated with HLS." https://gitlab.inria.fr/srokicki/Comet.

[2] N. Moreano, E. Borin, Cid de Souza, and G. Araujo, "Efficient datapath merging for partially reconfigurable architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 969–980, July 2005.

[3] "PicoRV32: A Size-Optimized RISC-V CPU.." https://github.com/cliffordwolf/picorv32. Clifford Wolf.

[4] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. 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. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The Rocket Chip Generator."