Removing Load-use dependencies bottleneck from CVA6 application class core

Gianmarco Ottavi[1], Davide Rossi[1], and Luca Benini[1,2]

[1]*DEI, University of Bologna, Italy*
[2]*IIS lab, ETH Zurich, Switzerland*

Abstract

Nowadays, RISC-V is one of the most widespread instruction set architectures in industrial and academic environments. The open-source nature of the ISA promotes the sharing of many designs that can be freely accessed online. In this work, we focused on CVA6 [ZB19] to perform an architecture exploration and optimization of the processor pipeline. CVA6 is an in order and single-issue application class core with 6 pipeline stages, maintained by the OpenHWGroup foundation. We established a performance baseline from which we decided the architectural changes presented in this work. We produced and overhauled a processor backend with the goal to optimize the load-use dependencies, resulting into IPC gain ranging from 5% to 29% on tests suffering from this problem.

1 Methodology and Results

With the aim to derive the required architectural changes, we established a performance baseline to understand where the current architecture was lacking. We analyzed two stages in the pipeline to identify the stall sources: one in the backend (issue stage) and one in the fronted (IF stage). For the latter, we checked whenever the instruction fifo that decouples the frontend from the backend was empty: stalls at this stage are mainly caused by flushes due to mispredicted branches. On the backend side, we added a set of signals to probe the cause of stalls on issuing the instruction to the execute stage (e.g., Functional Unit not ready, data hazard, structural Hazard). On each test, a report was printed with metadata showing the core stalls and types to visualize the core performance bottlenecks.

The benchmark suite utilized was embench-iot[1], a bare-metal set of tests covering a wide variety of algorithms for embedded platforms. Combining the added report mechanism explained earlier with the results of the benchmarks, we derived a couple of architectural zone to optimize: i) enhance the current branch predictor to reduce stalls on the frontend; ii) make architectural changes on the backend to alleviate load-use dependencies stalls. In this work, we decided to focus on ii), which required a complete rework of the backend. CVA6 presents four stages of pipeline on the backend: decode, issue, execute and commit stage. Depending on the instruction, the data-path in the ex-stage can require multiple cycles to complete (e.g., load takes 3 cycles in case of a hit). The in-flight instructions can complete out-of-order but are then committed in order. The structure used to achieve this goal is a *scoreboard* positioned in the issue-stage. Given the in-order issue of CVA6, instructions needs to be stalled whenever a load-dependencies is detected. The reworked backend presented in this work optimize this stall in the case of a hit in both TLB and Data Cache, which should be the most frequent cases.

Our work overhauled the backend targeting the issue-stage onward. We substituted the scoreboard with a fixed length pipeline where all integer instructions have the same completion time. The new integer execution pipeline is composed of 3 execute stages, which is the latency for a hit load operation. We added another pair of ALU and Branch Unit (BU) similar to the approach of the U74 from SiFive [U74]: a pair is placed in the first stage of the execution and the second on the last stage. This design allows the execution of integer instructions with a load-use dependency on the third pipe of the execution stage rather than stalling the instruction. The doubling of ALU and BU has a small cost in terms of area since both account for less than 1% of the total.

To keep track of the hardware costs, we run topological synthesis on both CVA6 and the core proposed in this work. The new architecture decreased the area of roughly 2% (the core area is dominated by the caches that account for 60% of the area). For what concerns timing, we had a regression of 4% as operand forwarding became more complex due to the elevated number of pipeline stages. The new backend brings no performance regression on tests which did not suffer from load-use dependencies while achieving 5 to 29% IPC gains on tests that heavily encountered load-use dependencies on integer instructions.

References

[U74] U74 core complex manual. page 34. https://starfivetech.com/uploads/u74mc_core_complex_manual_21G1.pdf.

[ZB19] F. Zaruba and L. Benini. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11):2629–2640, Nov 2019.

[1] https://github.com/embench/embench-iot