Design of the frontend for LEN5, a RISC-V Out-of-Order processor

Master's thesis summary

Candidate: Marco Andorno
Supervisor: prof. Maurizio Martina

December 2019

1 Introduction

Out-of-order processors have been the reference architectural paradigm for high performance computing since their introduction, as they provide the best ILP exploitation, despite the hardware complexity overhead. For this reason, the concept of the LEN5 project sparked from the interest in designing such a processor firsthand, to go one step further from the theory learnt during courses. LEN5 features a dynamically scheduled pipeline based on Tomasulo's algorithm, that, thanks to its distributed control approach, allows a completely modular execution pipeline, based on dedicated reservation stations before each functional unit, to provide operands and perform ou-of-order register renaming. Instructions are then put back in program order at the commit stage, by means of a reorder buffer (ROB).

The instruction set on which LEN5 is based is RISC-V, which is a comparatively new completely open source ISA, that in turn allows designing open source hardware cores. RISC-V is steadily growing in popularity both in academia and industry also thanks to its modularity, that is the fact that it offers a number of instruction set extensions that build upon the base ISA, allowing designers to include only some of them to tailor the architecture to their specific needs. In particular, LEN5 implements the RV64G instruction set, which includes integer, multiply and divide, floating point and atomic instructions.

This work in particular focuses on the frontend of LEN5, that is the part of the core responsible of generating addresses, predicting next directions and fetching instructions from memory to be issued to the execution stages.

2 Design

The design follows a modular approach, divided into two pipeline stages. The first one is the PC generation stage, in which the next fetch address is generated based on a list of priorities: exceptional behavior, misprediction, predicted branch target or PC+4 default assignment.

The second pipeline stage, the fetch stage, is responsible of receiving the current PC from the PC gen stage, determining if the instructions requires a cache access and eventually pushing the instruction to the issue queue, which separates the frontend from the later execution stages.

The Instruction Fetch Unit (IFU) contains two line registers where previously read cache lines are saved, in order to fetch consecutive instructions from them and to reduce the total number of memory accesses. A presence checker determines if the requested instruction is present in one of these registers and if that is not the case informs the fetch controller that a new cache read is needed. These requests are then sent to an instruction cache interface, which is in charge of handling the handshake signals to correctly send the address and receive data to and from the cache itself. Finally, an instruction selector chooses the selected instruction among the correct line source (i.e. either the cache output or one of the line registers) according to what the controller signals. All the operations of the IFU are controlled by a Mealy FSM, whose higher reactivity is needed to ensure a throughput of one instruction per clock cycle in the best case

The other significant block if the fetch stage is the Branch Prediction Unit (BPU). It features a gshare branch predictor and a Branch Target Buffer (BTB), which is a small direct mapped cache that stores the target address for taken branches only. When the predictor outputs taken and the BTB hits on the current address, then the branch is predicted taken. If it later gets discovered as a misprediction, both data structures are updated accordingly.

The gshare predictor contains a table of 2-bit saturating counters, which are simple FSMs that change prediction only after two consecutive misprediction, improving accuracy on loops. This table is indexed by a hash made by an XOR of the current PC and a global history register, in which past branch outcomes are shifted in when they are resolved. This way, both local and global information is retained when making a prediction.

Finally, for what concerns stalls, the IFU is able to handle cache misses and busy issue queue by pausing the internal fetch operation and to flush all its structures if a misprediction or exception occur. In this latter case, also BPU structures must be flushed.

3 Results

The entire design was verified using Verilator fo linting and ModelSim for simulation, with ad hoc testbenches. Then, it was synthesized under different conditions using Synopsys Design Compiler.

For what concerns the BPU, a software model written in C was used for benchmarking using four different branch traces, coming from the SPEC benchmark suite. A first analysis highlighted that there is no single best initialization value for the 2-bit counters, which will then be initialized to zero, that is the simplest value. Gshare accuracy simulations evidenced that a length of the global history register around 16 bits yielded the best accuracy values, with some test programs easily reaching beyond 99% accuracy. The BTB can only decrease this baseline accuracy because of misses, so other simulations were run to find the optimum value of the BTB index length for which this penalty becomes negligible, which was established to be around 12 bits.

Such large values for these two parameters result in a significant amount of area occupied by the BPU structures, as evidenced also during synthesis, which can be up to tens of kilobytes or easily over 1.5 million gates and a great part of the total frontend area. This of course must be taken into account when deciding on the tradeoff between accuracy and cost.

Even worse, the BTB decoding network also turns out to be the critical path of the entire design, limiting the maximum clock frequency the bigger the BTB gets. These are both unfortunate results, that however could be potentially solved by using a proper SRAM memory with optimized decoders and smaller footprint with respect to this simple register file implementation of such large structures.

Even if the results obtained are not the best one could hope and the performance of LEN5 cannot compete with state-of-the-art industrial processor, this work has fulfilled the initial objective, which was to explore the processor design flow and gain experience on the difficulties it presents. Moreover, this can be the initial starting point for the future development of a RISC-V based platform at our university, which could be of use both in teaching and research, by improving it and implementing different ISA extensions.