Hardware-Software Co-design for Low-cost AI processing in Space Processors

Design discussion for the LEON3 dual pipeline

Marc Solé I Bonet

Introduction:

In this report I will present and justify the different design decisions that were taken for the modification of the LEON3 integer pipeline into a dual issue pipeline converting the processor into a static superscalar processor with support for predication.

Duplicated hardware:

Many of the control signals of the processor will need to be duplicated, this mainly refers to the instruction and PC registers for each stage as well as additional signals that identify the operation and control the processor response.

Also the cache (instruction and data) will have their ports duplicated in order to serve two instructions at the same time as well as to allow for two simultaneous memory accesses. The same will be done with the register file having four read ports and two write ports.

The main integer pipeline will have it's ALU and additional components duplicated but not the special ones such as the multiplication unit or the AI SIMD module. Neither will be duplicated the branch and flow control logic.

Finally all bypasses will be extended to allow using the result produced by the operation on the other line. In this case the modifications will affect the MUL and SIMD modules' output and input signals.

Simultaneous instructions:

The LEON3 processor follows the SPARC v8 ISA, one important characteristic of this is that after a branch the following instruction is executed regardless of the condition result, this simplifies the implementation of dual pipeline as there is no need to cancel the other line instruction depending on the condition result of the branch. Nonetheless, there will be a need for correct managing for the two PC values to keep the coherence of the program.

With the duplication of the integer pipeline we have to consider the affectation on the dependencies of simultaneous instructions, this part should be optimized by the compiler to place the instructions at the optimal distance to maximize the performance. However, if two simultaneous instructions have a dependency, the receiver will be halted for the number of cycles necessary to have the result available. During this time, for simplification, in the other pipeline *nops* will be injected until the halted instruction proceeds.

Another possible situation is that both lines attempt to write the same register at the same time, in this case only the younger instruction will commit and the result for the previous one will be discarded, but not the possible exceptions it may trigger.

The same will be true for data accesses, in case both instructions access memory the address will be compared and actions will be taken depending on the operation. If both are writes, as in the previous example, the older instruction data will be discarded. If one the younger is a read and the older a write the data will be directly passed and the store will occur at the same time. Finally, if the situation is the

other way around, ideally it will be read first and later the value will be written. If this is not possible the write will be halted until the read commits. Both lines will halt, only the read allowing to proceed.

Lines distribution:

As said before some of the logic for the different instructions will be exclusive to a single line, that means that the instruction will only be issued, maintaining the program order, if the line with the appropriate hardware is available.

However, a constraint that will be kept is that always the instruction on the line 1 will be older than that of the lane 2. With this in mind, and going back to the previous sections, we can see some considerations, like the need for enforcing program control instructions to execute in the first line. With this we can execute the following instruction simultaneously.

On the other hand, multiplication and SIMD instructions will go to the second line, and if needed *nop* instructions will be passed to the first one.

Summarizing:

- **Line 1**: ALU operations, memory accesses and program flow control instructions
- **Line 2:** ALU operations, memory accesses, SIMD module instructions and multiplication module instructions

Predication:

At the moment the predication modifications are left until the dual-pipeline is working appropiately. The necessary changes include the setting of the predicative value, and the conditional execution of the instructions regarding on it’s value.