POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Design and verification of a RISC-V superscalar CPU

Laurea Magistrale in Electronics Engineering - Ingegneria Elettronica

Author: Marco Vitali and Sebastiano Vittoria

Advisor: Prof. Davide Zoni

Co-advisor: Dr. Andrea Galimberti

Academic year: 2022-2023

1. Introduction

In 1965, Gordon Moore made an influential observation that has had a significant impact on modern computing. It is now widely known as Moore's law and it states that the number of transistors on a microchip will double approximately every two years. This trajectory was further supported by Robert Dennard's scaling principles, which predicted that as transistors shrink in size, their power density would remain constant, leading to a proportional increase in performance. The RISC-V Instruction Set Architecture [5] is widely adopted in academic research due to its open-source nature. Its modularity and extendibility allow for a high degree of customization in terms of ISA extensions or even the creation of brand-new instructions, thanks to the free spaces left in the opcodes list. The simpler nature of a RISC-V is also optimal for the development of more logically complex pipelines, which boosts performance in exchange for a minor increase in area and power consumption. One such approach is to allow the concurrent propagation of more than one instruction at a time through the whole pipeline, creating the distinction between scalar and superscalar CPUs. Processor design has long focused on the dichotomy between Complex Instruction Set Computing (CISC) and Reduced Instruction Set Computing (RISC) architectures. CISC architectures aim to reduce the number of instructions per program, theoretically leading to more efficient code execution, by using rich instruction sets. On the other hand, RISC architectures streamline instruction sets, emphasizing simplicity and efficiency in execution. Superscalar architectures represent a paradigm shift from the traditional scalar architectures. A superscalar CPU presents more complex logic networks and, as a consequence, a higher area occupation and power consumption but with the advantage of completing, ideally, multiple per clock cycle. If designed carefully, this type of CPU can represent a good middle ground between low-cost and high-performance solutions. Often, FPGAs represent the best target for the development of a design, thanks to their re-programmability, consequentially reducing the time to market of a product. Noticeable examples of RISC-V superscalar CPUs are Frontgrade Gaisler's NOEL-V [4], an in-order dual-issue CPU, and UC Berkeley's BOOM, superscalar and capable of out-of-order execution [3].

This thesis proposes a description of ViVit, a RISC-V superscalar in-order dual-issue CPU, and its verification method exploiting a customized version of an open-source ISA simulator.

The three main contributions are:

- The design of the CPU's microarchitecture. The present study centers on the microarchitecture design of the superscalar CPU, with a particular emphasis on each pipeline stage. Additionally, we analyze which pipeline stage could become the critical path of the stage.
- The verification infrastructure consists of a functional ISA simulator and a socket for communication. The custom features and their usefulness are discussed, along with an in-depth methodology for verification.
- Experimental evaluation. To evaluate the quality of a CPU, its performance, power, and area results are analyzed across all the extracted data. This analysis provides an overall view of the CPU's quality and particularly focuses on identifying the stages that have a significant impact on power consumption and area occupation.

2. Methodology

Hardware design poses numerous challenges, primarily due to the concurrent nature of its logic networks and the presence of registers that require synchronization between blocks. When selecting an FPGA board, system resource utilization is a key metric, as higher data volumes result in increased area occupation. It's important to note that transitioning from a scalar to a superscalar architecture substantially increases area occupation, as most ports and interconnections must be duplicated to support multiple instructions propagating through the pipeline. Additionally, logic networks contribute significantly to this since they must be extended, often doubling the initial complexity due to possible dependencies between multiple instructions.

2.1. Architecture of the superscalar CPU

The presented superscalar architecture features a 7-stage dual-issue pipeline, composed of the Instruction Memory, Fetch, Decode, Issue, Execute, Reorder Buffer, and Register File stages. It supports in-order dispatch of instructions and out-of-order completion.

The **Instruction Memory (IM)** of the architecture is implemented as a RAM, accessible from the Fetch stage through the desired Program Counter. A relevant feature for a superscalar Architecture is a Pre-Fetch logic, which accounts for a set of particular limit cases.

The **Instruction Fetch (IF)** stage is the second stage of the pipeline and datapath. It's responsible for extracting two instructions from the corresponding memory in one clock cycle. Additionally, it can perform a "pre-decode" of the input instructions, which helps synchronize the front end blocks, especially in the case of subsequent bundles of instructions containing branches. At each clock cycle, a bundle of two instructions is extracted from the Instruction Memory and fed to the pre-decode logic. The pre-decode logic

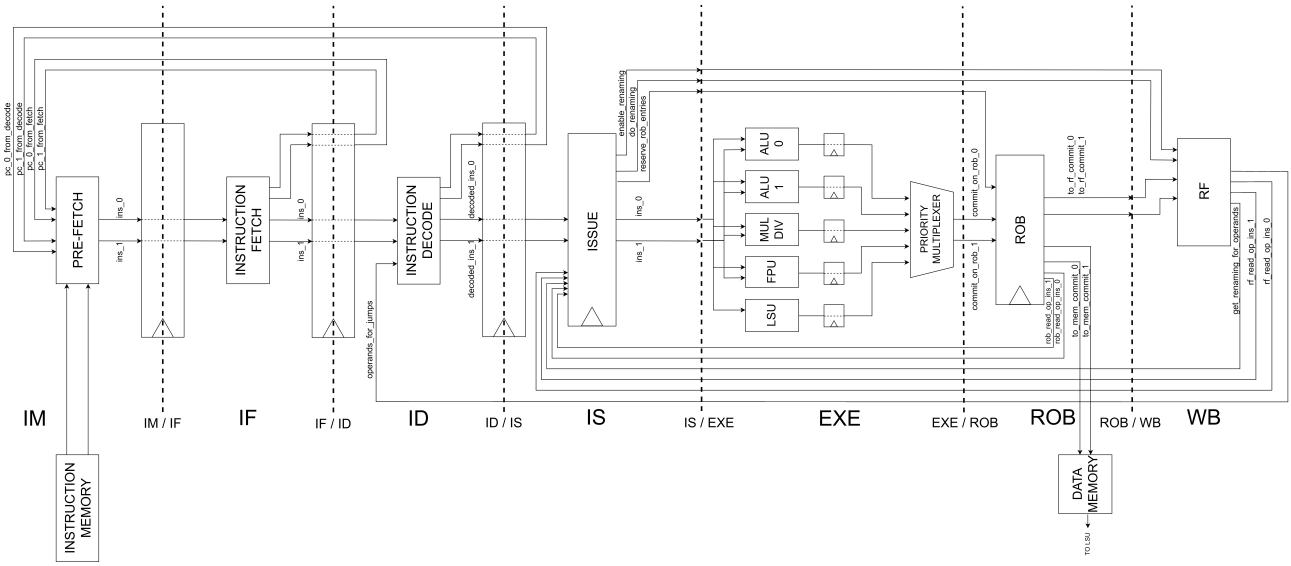

Figure 1: Microarchitecture of ViVit, the proposed superscalar CPU

distinguishes between three cases: the first instruction is a conditional or unconditional jump, the second one is a jump, or neither are jump instructions.
The next stage is the superscalar **Instruction Decode (ID)** stage. This stage supports the decoding of 32-bit RISC-V instructions coming from the I-, M-, and F-extensions. It has two main duties: instruction decoding and jump resolution and calculation. The Decode logic is implemented as a sequential positive edge-sensitive net. The main challenge lies in the jump logic, as a set of ports must be created to propagate a read address to the RF and receive from it the status of the requested register and its value. This is done through combinatorial logic, which implements an asynchronous inter-module communication capable of resolving the read request without waiting for a rising edge of the clock. However, the high weight of the logic results in a limitation on the maximum achievable frequency.
After a bundle of instructions is decoded, it is propagated towards the **Issue (IS)** stage, which is responsible for dispatching the instructions to the Execute stage and renaming the operands to guarantee the correct out-of-order execution and completion of the operations. Each instruction passes first through the fill queue logic, which saves it into a FIFO-like memory called issue queue. Then, the instruction operands are read through the double read operands logic, and finally gets dispatched.
The **Execute (EXE)** stage is in charge of producing the result values. It is composed of a parametric number of functional units that can be chosen at design time based on the application needs. Having a double dispatch means that every type of functional unit needs a queue in case a bundle is composed of two instructions that need to be executed on the same unit. After a result is produced by the Execute stage, the commit on the Reorder Buffer is managed by a Priority Multiplexer, which accelerates the propagation of load and stores.
The **Reorder Buffer (ROB)** is a FIFO-like structure with a parametric depth and entries composed of multiple fields. The main role of the Reorder Buffer is to commit the results of completed instructions, granting the correctness of the program order even in an out-of-order completion architecture. So, it's the duty of the ROB itself to produce the result and additional information of the one or two head instructions inside it. Multiple commit routers guide the committing instructions towards either the Register File or the Data Memory. There is also a combinatorial logic used to send the value requested by the double read operands in the Issue stage. This logic must be realized in a combinatorial way to get the value at the same clock cycle the data is required.
The last pipeline stage of the superscalar CPU is the **Register File (RF)**, which stores all the register values required by the ISA. Sequential logic is added to take care of both doing and undoing the renaming of the register.

2.2. Verification of the superscalar CPU

To ensure that the architecture described in the previous section is executed correctly, the chosen verification approach was to monitor every instruction commit. To automatically verify the execution during a simulation, the ISA simulator and Vivado had to communicate with each other. They were connected through a socket to send commands from the testbench and receive data to perform the verification routine. The ISA simulator adopted is Spike, the RISC-V golden model for verification.
To perform the verification, Spike must be started in interactive debug mode and informed which application to run. It waits for a connection with the client. Then, in the testbench, the architectural instruction and data memory are initialized with the content of the application, and then the

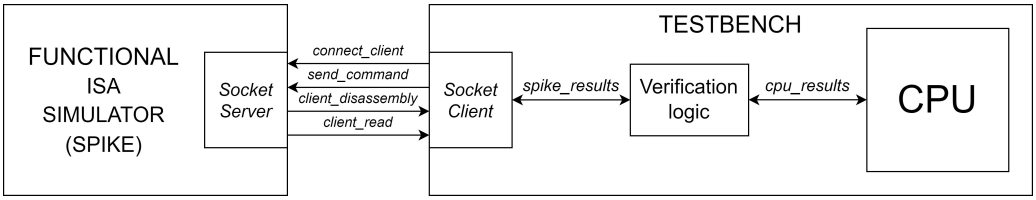

Figure 2: Scheme of the Verification process

client-side connection is established.
After the architecture simulates one clock cycle, the TB verifies if one or two new commits occurred. If they did, it instructs Spike to simulate one step through a SV function. Then, it compares the PC, destination, and result coming from the golden model with the ones produced by the architecture. If a mismatch occurs in any of them, the simulation stops. In case of a store instruction, the TB calls the *spike_mem* SV function, which exposes the content of the memory address that was just written in the functional simulator. This is necessary since the commit log for stores is not automatically produced. Finally, the TB checks if the committed PC matches a target final PC, which is a parameter set according to the dump of the executed application. If the two match, the simulation ends, and the verification is successful.

3. Experimental evaluation

The proposed architecture is implemented to validate it and evaluate a set of quality metrics. The design is written in SystemVerilog. Synthesis and implementation are carried out on Vivado 2022.2, targeting an AMD Xilinx Artix-7 75 FPGA, which provides 47200 Lookup Tables (LUTs), 94400 Flip-Flops (FFs), 105 BlockRAMs (BRAMs) and 180 Digital Signal Processing units (DSPs). The target frequency of the implementation is 77 MHz, using the default directives. For the verification, the used Spike version is 1.0.0. Everything is hosted by a machine running Linux OS distribution Ubuntu 22.04. All the bare-metal *elf* executables were compiled using the *riscv32-unknown-elf-gcc* compiler based on the GNU RISC-V 32-bit Toolchain [1] and the linker provided by the official *riscv-test* repository or WCET website [2]. Regarding the interface between Vivado and Spike, DPI (Direct Programming Interface) is used and all the C functions are compiled through the *xsc* compiler. For the evaluation, the chosen configuration of the pipeline parameters is the following: RV32I base ISA with M and F extensions, 2 ALUs with one cycle of latency, 1 MULDIV with 5 cycles of latency, 1 FPU with 5 cycles of latency, 1 LSU with 3 cycles of latency, 16 entries deep issue queue, 64 entries deep ROB, 32 bit wide and 2^{10} entries deep Instruction Memory, 32 bit wide and 2^{16} entries deep functional Data Memory. Every FU is functional. For the rest of this Chapter, the configuration of the architecture described is referred to as ViVit. Regarding the results, using Vivado's post-implementation functionality *Report Power Utilization* on the final CPU design, a hierarchical view highlighting the power consumption of every module can be obtained, while the quality metrics such as resource utilization and timing must be extracted after Vivado's synthesis and implementation. Based on the WCET benchmarks [**?**], we extracted the following information: the total number of commits, the number of single and double commits, and the percentage of double commits over the total. This percentage is related to the superscalarity, which is the ability of a processor to execute multiple instructions in parallel.

Table 1: Numbers of commits and percentage of exploitation of the superscalar properties

WCET	Total	Double	Ratio
bsort100	1237	310	50,1%
cnt	2006	567	56,5%
crc	21186	4600	43,4%
fdct	1363	363	52,3%
jfdctint	1748	444	50,8%
prime	1754	3	0,3%
select	845	156	36,9%
fac	124	24	38,7%
janne_complex	77	10	25,9%
lcdnum	101	11	21,7%
matmult	12156	4422	72,7%

As shown in Table 1, the architecture is capable of exploiting its superscalar features in almost all benchmarks, except for the *prime* benchmark which has nested data dependencies and jumps. This hinders the use of dual-issue and dual-commit functionality, highlighting that a superscalar architecture may not be the best solution if there is not much exploitable instruction-level parallelism in the program code. Regarding the *prime* WCET benchmark, only 3x2 instructions take advantage of the double commit feature. However, in the case of the *matmult* WCET benchmark, the percentage of instructions that

exploit the double commit feature is the highest among all benchmarks, with a percentage of 72.7%. This is due to the fact that the code is highly compatible with a dual-issue architecture, and hence allows for the creation of independent instruction bundles in most cases. This benchmark provides an ideal scenario for a dual-issue CPU to leverage its superscalar feature, resulting in the best percentage among all reported applications. The remaining benchmarks have an average exploitation of the dual-issue and dual-commit features at around 47%. Thus, we can conclude that the architecture is capable of fully exploiting the dual-issue and dual-commit features almost half of the time. The percentages are referred to the total CPU power budget.

Table 2: Power estimation after implementation. The percentages are referred to the total CPU power budget.

Module	Total [mW]	Ratio
CPU	**251**	
Fetch	1	0,4%
Decode	4	1,6%
Issue	46	18,4%
ROB	160	63,9%
RF	40	15,7%

Looking at Table 2, it can be seen that the Reorder Buffer represents the module with the highest power consumption with a value of 160 mW, which means that 63,9% of the total power budget is due to this stage. This is mainly due to the Reorder Buffer structure itself, whose entries are composed of different fields, each of them accessible from different logic networks, like the Fill Queue of the Issue stage writing destination, Program Counter and store amount, or the Execute stage committing a result, or again the Issue accessing these results for the double read operands logic. The commit on ROB and commit from ROB, including the Commit Router, do not influence significantly the power metric. The combinatorial network used for the double read operands represents another important contribution to this value. The Register File presents an average power consumption of 40 mW, which again is mainly due to the combinatorial network used by the Issue stage, but also to the renaming logic which must perform many nested checks to avoid problems in limit cases where an instruction commits on a register which is currently being renamed. The power consumption of the Issue stage, 46 mW, is highly impacted by the double read operands logic and its correlated combinatorial networks. Finally, the Fetch and the Decode stages do not contribute heavily, with a combined 5 mW of total power consumption. This low value is mainly due to the fact that the logic nets implemented in these stages are extended but simple.

Table 3: Resource and Timing report after implementation. The target frequency is 77 MHz.

Module	LUT	FF	WNS
CPU	**26298**	**13593**	0.594 ns
Fetch	299	322	6.196 ns
Decode	973	302	5.940 ns
Issue	2524	2134	1.626 ns
ROB	16204	9491	2.729 ns
RF	6233	1280	3.962 ns

The table above shows all the results, with a focus on the percentage of utilization of the target FPGA. We first extracted the number of overall CPU resources used, and then we implementated each module to collect the same set of data. We obtained the Worst Negative Slack (WNS) of each module with the same clock frequency of the CPU, which is reported in the *Fmax* column. Looking at Table 3, we can see that the stage with the highest number of used LUTs is ROB. This is mainly due to the two combinatorial networks used to communicate with the Issue stage: one to look for the values during the read operands, and the other to insert PC, destination, and store amount during the fill queue. These networks are particularly complex, as they must be able to access every cell of the ROB in a single clock cycle without a proper synchronous read port. Regarding the FFs utilization, it is mainly due to the ROB structure itself, since every entry is composed of many different fields. A similar argument applies to the Issue stage, where a portion of the FFs is used to implement the issue queue. The number of LUTs is justified

by the presence of three heavy combinatorial networks: one used to gather operands from RF and ROB, one to perform and enable the renaming in the extension fields of the Register File, and the last one to implement the double read operands logic itself. Regarding the RF, FFs are mainly used to implement the Register File, while the number of LUTs is not due to the logic used for the commit of the instructions, but rather to the net used to perform enabling and disabling of the renaming. In terms of timing, the CPU can run at a maximum frequency of 77 MHz, having a WNS of 0.594 ns. The Issue module represents the critical path for the architecture, having a large combinatorial net that limits the timing performance of the stage, with a WNS of 1.626 ns. The total CPU WNS is much lower than the one of the limiting stage since it considers the totality of the interconnections between blocks. In this case, the critical path becomes the combinatorial networks for the double read operands, which are not taken into consideration during the implementation of the single module.

4. Conclusions

This thesis explores a superscalar RISC-V dual-issue CPU, highlighting the intricacies and potential advancements in modern processor design. Investigating the architecture's ability to simultaneously execute two instructions per clock cycle, has not only showcased the efficiency gains but also underscored the significance of adopting the RISC-V instruction set architecture. The ViVit CPU emerges as a promising avenue for achieving higher throughput and addressing the growing demands of computational workloads, with its custom verification tool and infrastructure. It aims to propose a solution to a specific market need for a high-performance processor with a medium-low area occupation and power consumption. Looking ahead, there are several avenues for future improvement. One key area of focus is the hardware implementation of the Execute stage and the refinement of compiler technologies to better exploit the dual-issue capabilities. Additionally, exploring advanced branch prediction techniques and enhancing the instruction fetch mechanism can contribute to reducing pipeline stalls and improving overall execution efficiency. Moreover, the integration of more sophisticated out-of-order execution mechanisms holds the potential to unlock greater parallelism within the processor, leading to improved performance in diverse computational workloads. Power efficiency and scalability should be at the forefront of future developments. Exploring techniques for dynamic voltage and frequency scaling, as well as investigating advanced power gating strategies, can contribute to creating more energy-efficient processors without compromising performance. In summary, the continuous evolution of the superscalar CPUs necessitates ongoing research and development efforts. Future improvements should address both hardware and software aspects, to unlock the full potential of this architecture and meet the ever-growing demands of modern computing.

References

[1] Homepage of GNU RISC-V Toolchain Repository: https://github.com/riscv-collab/riscv-gnu-toolchain.

[2] Homepage of Mälardalen WCET benchmarks: http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

[3] Alexander Dörflinger, Mark Albers, Benedikt Kleinbeck, Yejun Guan, Harald Michalik, Raphael Klink, Christopher Blochwitz, Anouar Nechi, and Mladen Berekovic. A comparative survey of open-source application-class risc-v processor implementations. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, CF '21, page 12–20, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Manoj Sharma, Ekansh Bhatnagar, Kartik Puri, Amitav Mitra, and Jatin Agarwal. A survey of risc-v cpu for iot applications. In *Proceedings of the International Conference on Innovative Computing & Communication (ICICC) 2022*, February 2022. Available at SSRN: `https://ssrn.com/abstract=4033491` or `http://dx.doi.org/10.2139/ssrn.4033491`.

[5] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. The risc-v instruction set manual, volume i: User-level isa, version 2.0. Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley, May 2014.