Report 4: Synthesis and Project of a Simplified ALU for Monocycle MIPS

Bruno Bianchi Pagani, Cid Fernando Sette de Borba, Gabriel Salmoria,
Guilherme da Silva Pierri, Gustavo Nunes Viana, João Gabriel Feres
Dept. of Informatics and Statistics (INE), Federal University of Santa Catarina (UFSC), Florianópolis, Brazil
{brunobianchipagani, cidfernando, gabrielsamoria1, guilhermepierri8, nunesvianagustavo, joaogabrielferes}@gmail.com

***Abstract*—This project presents the design and synthesis of a simplified Arithmetic Logic Unit (ALU) for the MIPS architecture, incorporating multiplication functionality. The focus is on creating an efficient and compact ALU while maintaining compatibility with the MIPS instruction set. The design encompasses key elements such as the multiplier, addition, subtraction, set less than, and bit-wise AND and OR operations. Emphasis is placed on achieving simplicity in the context of MIPS architecture. The synthesis process involves optimizing the ALU design for practicality, ensuring a pragmatical and efficient implementation while test benching all operations for viability and trust enforcement. This project contributes to the broader goal of understanding of the computational capabilities of MIPS processors, particularly in improving comprehension over the Monocycle MIPS potential and usage, all while adhering to the principles of simplicity and functionality.**

***Index Terms*—ALU, MIPS, VHDL, HDL, Computational Capabilities, Hardware, Multiplication, bit-wise**

I. Introduction

In the ever-evolving realm of processor design, the importance of a versatile Arithmetic Logic Unit (ULA) cannot be overstated. This project delves into the intricacies of developing a sophisticated ULA using VHDL, tailored for the MIPS architecture, with a specific emphasis on not only fundamental arithmetic operations like addition and subtraction but also on crucial logical OR, AND and slt operations.

The MIPS architecture, renowned for its simplicity and elegance, serves as the ideal platform for this endeavor. Beyond the conventional arithmetic tasks, our ULA is engineered to seamlessly execute logical operations such as AND and OR, providing the processor with powerful tools for data manipulation and decision-making. Additionally, the integration of relational operations like less than further expands the ULA's utility in branching and conditional execution scenarios, turning ULA's hardware description even more similar to the physical one.

This introduction sets the stage for a detailed exploration of each facet of the ULA's VHDL design. The subsequent sections will delve into the nuanced architecture and control logic required to facilitate not only addition, subtraction and multiplication but also AND, OR and less than operations. By enhancing the MIPS architecture with these diverse capabilities, our goal is to improve the understanding of how an ULA works on the Monocycle MIPS and how to implement its operations.

II. Operations

A. Multiplication

The multiplication operation within our designed Arithmetic Logic Unit ULA was crafted using VHDL (VHSIC Hardware Description Language), which allows for a systematic and modular representation of the multiplication algorithm, ensuring clarity and maintainability in the design process. One noteworthy aspect of our approach is the precision assurance strategy embedded in the multiplication process. To guarantee accurate and reliable results, a precision of 65 bits was implemented. This precision level not only accommodates a broad range of numerical values but also mitigates potential issues related to overflow or loss of significant digits. By adopting this strategy our multiplication operation ensures that we have trustful signed multiplication for MIPS's 32 bits words what makes the precision even better.

B. Addition

The addition operation within our ULA is executed with a degree of parallelism through the implementation of 32 full adders. Each full adder, a fundamental building block in digital arithmetic circuits, incorporates three inputs - two operands and a carry-in from the previous bit - producing a sum and a carry-out. The parallel structure of 32 full adders allows for the simultaneous processing of multiple bits. This approach prefers simplicity but also ensures the calculations to be trustful while computing all possible values.

C. Subtraction

The subtraction operation in our ULA is accomplished with a meticulous design that leverages the inherent duality between addition and subtraction. Rather than implementing a separate subtraction circuit, our approach involves utilizing the same 32 full adders employed for addition, while incorporating a complement operation. By complementing the subtrahend and setting the initial carry-in to 1, the subtraction process effectively becomes an addition operation with adjusted inputs. This designs helps to use less circuit area and reutilize the same components used for summation.

D. OR Operation

The bitwise OR operation stands out as a pivotal component, addressing various computational needs. Firstly, it plays a crucial role in data masking and filtering, allowing for the selective modification of specific bits or fields within binary representations. Additionally, the bitwise OR operation on MIPS is fundamental for flag and state checking in system programming. This operation is instrumental in setting and verifying flags and states, thereby influencing program flow, managing errors, and controlling various system states. Moreover, in the context of MIPS digital signal processing, the bitwise OR operation is indispensable. It facilitates the combination or filtering of signals based on specific conditions, contributing to tasks such as audio processing, image manipulation, and telecommunications. This process is simply done by doing OR operation for each bit in two different words.

E. AND Operation

The logical AND operation finds significant applications in diverse computational scenarios. Firstly, within the realm of data processing such as its usability on simple circuit. This operation allows for the precise extraction of relevant information or the application of specific conditions to datasets, contributing to efficient data manipulation. Moreover, in security mechanisms and access control systems on MIPS, the logical AND operation plays a interesting role, enabling the combination of multiple access conditions, ensuring that access is granted only when all specified criteria are simultaneously met. This process is simply done by doing AND operation for each bit in two different words.

F. SLT Operation

The Set Less Than (SLT) operation is a fundamental element for making numerical comparisons, enabling processors to assess whether one operand is less than another as it is named after. This operation is executed by subtracting the second operand from the first and evaluating the result to determine its signal. If the result is negative, the SLT operation the out signal is 1, indicating that the first operand is indeed less than the second; otherwise, it sets the register or flag to 0. The SLT operation develops an important paper while conditional branching. This operation is extensively used in algorithmic decision-making, loop control, and other scenarios where numerical relationships between operands influence program flow within the MIPS architecture.

III. WORKING

The architecture of the Arithmetic Logic Unit (ALU) for a MIPS processor, incorporating multiplication operations, is meticulously designed to perform arithmetic and logic operations. The overall structure consists of two main interrelated components: the control block and the operative block.

The control block plays a vital role in coordinating the ALU operations determined by the selector. It receives two fundamental inputs: the operation code (ULAop), with 2 bits, and the function (funct), with 6 bits. The interpretation of these signals determines the overall behavior of the system, generating the 3-bit selector that will guide the operative block. An operation code ”00” indicates an addition operation, ”01” refers to subtraction, ”10” requires a specific interpretation of the function, while ”11” indicates a multiplication operation. This flexibility allows for the efficient execution of various operations, aligning with the intrinsic versatility of the MIPS architecture.

The operative block, responsible for the effective execution of operations, is instantiated with 32-bit inputs (inA and inB) and a 65-bit output (OutS). Operations are driven by the 3-bit selector, whose individual bits have distinct roles. The least significant bit (selector(2)) determines whether the operation will be an addition or subtraction. If ”1”, subtraction occurs; if ”0”, addition occurs. The most significant bit (selector(0)) influences the choice between an AND or OR operation. If ”1”, the operation will be an OR; if ”0”, an AND. The middle bit (selector(1)) drives the multiplexers, influencing the final choice between the various available operations. This level of granularity in the selector allows for precise adaptation to the nuances of machine language instructions.

Multiplexers, or muxes, are essential elements in the architecture, playing a crucial role in selecting specific results from operations. The selector is instrumental in directing the flow of data, ensuring the correct execution of instructions.

The inclusion of the multiplication operation significantly expands the computational capability of the MIPS processor. This operation is performed by a dedicated multiplication unit, standing out as an innovative functionality, and perhaps unnecessary for this level, that performs complex calculations.

In summary, the architecture of the ALU for the MIPS processor with multiplication operations demonstrates a careful integration of components, providing a solid foundation for efficient and flexible instruction processing. This approach represents a substantial contribution to the evolution of processor architectures, especially in contexts that demand sophisticated and versatile computational power.

IV. TESTBENCHING

The development of self-testable testbenches is crucial to ensure the robustness and reliability of complex digital systems, such as the Arithmetic Logic Unit (ALU) mentioned in the context of the MIPS architecture. Efficient testbenches are designed to verify the correct operation of the Design Under Test (DUT), which in this case is the ALU. An advanced approach to test vector generation involves using a reference model, known as a golden model, representing an expected and reliable behavior of the DUT.

The DUT, in this case, is the MIPS ALU, composed of the operative and control blocks. When developing a self-testable testbench, it is essential to consider the various operations supported by the ALU, including addition, subtraction, AND, OR, SLT, and critically, multiplication. The testbench should be capable of generating varied and complex inputs, representative of real operating conditions.

The approach with a golden model involves having a highly reliable reference model, which can be a simulator or a software implementation performing the same operations as the DUT. This model is used to generate sets of test vectors known as reference test cases. These test cases are then employed to verify whether the DUT is producing consistent results in accordance with expectations.

When running the testbench, inputs are applied to the DUT, and the results are compared with the corresponding results generated by the golden model. Any discrepancies between the results of the DUT and the golden model indicate potential issues in the design or implementation of the ALU.

Test vectors generated by the golden model should encompass a variety of situations, including extreme values and cases involving multiplication operations. This ensures that the DUT is tested in different scenarios, identifying potential weaknesses and ensuring its reliability in various operating conditions.

Creating a self-testable testbench with test vectors generated by a golden model is an effective practice for the comprehensive validation of the MIPS ALU. This approach not only verifies the correct execution of basic operations but also ensures that more advanced functionalities, such as multiplication, are tested comprehensively. This rigorous validation is essential to ensure that the MIPS processor is robust and capable of handling a wide range of instructions in real-world scenarios.

A. Golden Model

The golden model generator for different modules within the context of a MIPS processor's Arithmetic Logic Unit (ALU). The script offers functionalities to generate test stimuli for modules such as addition and subtraction (addsub), multiplication (mul), set-on-less-than (slt), and bitwise AND/OR operations (and$_{o}r$).

To use the golden model, users can execute the script with a specified option, representing the desired module. For instance, running python3 golden-model.py "addsub" generates stimuli for addition and subtraction, while python3 golden-model.py "mul" produces stimuli for multiplication. The option "all" generates stimuli for all modules, and "clean" removes previously generated stimulus files.

The generated test stimuli files facilitate comprehensive testing, covering a range of scenarios with random inputs for each module. These stimuli can be employed by integrating the corresponding module into the toplevel of the MIPS processor's design and selecting the relevant testbench for validation. This approach ensures thorough testing of the ALU's functionality across different operations, contributing to the overall reliability of the MIPS processor.

V. SYNTHESIS RESULTS

The synthesis results of the Arithmetic Logic Unit (ALU) for the Monocycle MIPS processor reveal critical parameters and crucial statistics to assess the performance and efficiency of the design. The identified critical delay was 32.907 nanoseconds, an essential metric to determine the circuit's latency. This critical delay can be attributed to specific conditions, such as falling to rising transitions in the "fnct[1]" input signal, resulting in a transition in the "s[29]" output signal. The analysis of these timings is vital to ensure that the circuit meets the necessary timing requirements for correct instruction processing.

Additionally, the board occupancy was recorded at 44%, indicating the proportion of FPGA resources used by the design. This value provides insights into the efficiency of using the available logic elements in the FPGA. Notably, only 9% of the logic elements were utilized, indicating efficient utilization of available hardware resources. As evidence of this efficiency, no registers were used.

The input and output parameters of the ALU are also crucial for evaluating its practical utility. The "a" and "b" inputs are both 32 bits, representing values used in arithmetic and logic operations. The "fnct" signal, with 6 bits, is another crucial input that determines the desired function of the ALU. The output "s" is 65 bits, covering a wide range of values resulting from the operations performed by the ALU.

In summary, the synthesis results provide a comprehensive view of the performance of the Arithmetic Logic Unit designed for the Monocycle MIPS processor.

VI. CONCLUSION

In conclusion, the synthesis and design of a simplified Arithmetic Logic Unit (ALU) for a Monocycle MIPS processor, as presented in this report, represent a significant step towards enhancing the computational capabilities of MIPS architecture. By incorporating fundamental arithmetic operations such as addition, subtraction, and multiplication, along with critical logical operations like AND, OR, and set less than (SLT), the designed ALU stands as a versatile and efficient component.

The detailed exploration of each operation, from the precision-ensured multiplication to the parallelized addition and cleverly implemented subtraction, demonstrates a commitment to simplicity without compromising functionality. The bitwise AND and OR operations, as well as the SLT operation, add further depth to the ALU's capabilities, addressing a wide array of computational needs within the MIPS architecture.

The meticulous architecture of the ALU, with its control block and operative block, allows for precise coordination of operations. The inclusion of a dedicated multiplication unit showcases an innovative approach, potentially extending the processor's computational power.

The testbenching strategy, utilizing self-testable testbenches with vectors generated by a golden model, ensures the robustness and reliability of the ALU. Testing across various scenarios, including extreme values and multiplication operations, contributes to comprehensive validation, affirming the ALU's functionality in diverse operating conditions.

In summary, this project contributes to a deeper understanding of the Monocycle MIPS processor's capabilities and usage. The emphasis on simplicity, functionality, and comprehensive testing aligns with the principles of efficient processor design. The designed ALU, with its integration into the MIPS architecture, represents a noteworthy advancement in the realm of processor architectures, offering adaptability and reliability for a wide range of computational tasks.

References

[1] PATTERSON, David A.; HENNESSY, John L. “Computer Organization and Design: the hardware/software Interface”, 3rd edition, Morgan Kaufmann Publishers, San Francisco, California, USA, 2007