**Team members:**

Bardan Bianca

Golya Carmen

Hondola Paul

Iordachescu Vlad

**Semigroup**: 2.1, 2nd year, Fac. Of Automation and Computers, specialization-Computers

***ALU Documentation.***

1. **Project overview and objectives.**

This project focuses on the design, structural implementation, and simulation of an 8-bit Arithmetic Logic Unit(ALU) using a Hardware Description Language(HDL). The ALU is capable of performing its classic operations, such as addition and substraction, but also multiplication and division. The design is fully structural, meaning that it is build using interconnected hardware components such as adders, multiplexers, control logic, and registers, rather than behavioural constructs.

A dedicated Control Unit is responsible for generating the control signals necessary to select and coordinate the arithmetic operations accordingly. For multiplication, the Booth Radix-4 algorithm has been chosen due to its efficiency in handling signed multiplication. For division, the fast division SRT-2 algorithm is implemented because it allows a faster and more hardware-efficient implementation compared to traditional restoring or non-restoring methods.

The main objectives of the project are as follows:

* To design a structurally organized 8-bit ALU using modular HDL components.
* To implement efficient signed multiplication and division using Booth Radix-4 and SRT-2 algorithms, respectively.
* To build a Control Unit capable of coordinating the data flow and operations selection via control signals.
* To validate the ALU’s functionality through simulation with testbenches and waveform analysis.
* To strengthen knowledge of digital hardware design, with focus on structural modeling, modularity, and algorithm-driven arithmetic units.

This project reinforces the practical understanding of arithmetic unit construction and the principles of control signal management in complex digital systems.

1. **Architecture diagrams and explanations.**
   1. *Top-level architecture:*

The 8-bit ALU si composed of several modular components, interconnected structurally. The main inputs, outputs, and modules are outlined below as follows:

1. Inputs:

* Clock, Reset, Begin;
* 8-bit Inbus.

1. Outputs:

* End;
* 8-bit Outbus.

1. Modules:

* ALU: implements addition, substraction, multiplication, and division.
* Control Unit: handles control signals based off the operation selected to be done.
* Multiplexer: selects the desired output, based on control signals.
* Register: stores input operands and intermediate values, if necessary.

The overall structure ensures modularity, making it easy to integrate, simulate and debug.

All modules are implemented in the attached .v files, along with the hardware platform configuration, further detailed both here and in the respective file.

* 1. *Control Unit overview:*

The Control Unit is implemented as a Finite State Machine(FSM) consisting of 17 distinct states, using a one-hot encoding method. Each state corresponds to a specific control phase of the ALU operation, such as operand loading, arithmetic processing, and result output, with each state represented by a dedicated flip-flop.

The FSM transitions between states are synchronized with the system clock and are triggered based on the current instruction type and internal status signals. The control sequence starts from an initial state(idle), proceeds through operand fetching, executes the ALU operation, and ends with writing the result to the output register.

These 17 states cand be categorized into functional groups:

* Initialization/reset;
* Operand load;
* ALU operation selection and execution;
* Result storage and output enable.

One-hot encoding was chosen for its simplicity in decoding logic, allowing each state to be uniquely represented by activating a single flip-flop. This approach simplifies control signal generation at the expense of increased flip-flop usage, which was acceptable given the limited number of states.

Each state in the FSM activates a specific set of control signals used to orchestrate data transfers, enable/disable registers, and trigger ALU operations. These signals are hardwired based on the current active state, ensuring predctible behavior.

* 1. *Arithmetic Unit breakdown:*

Within the Arithmetic Unit, there are several key components:

1. Adder/Substracter module: represented by a Ripple Carry Adder. This serves as a substractor as well, based on the way the carry-in is implemented: if it is 0, then it will execute an addition, else a substraction will be expected.
2. Multiplication: signed multiplication is pwrformed within the ALU, using Booth’s Radix-4 algorithm.
3. Division: division is handled also within the ALU using SRT-2 algorithm.

Control Unit activates each of these operations, depending on the operation code it’s given. It interprets the op\_code vector and activates the corresponding logic in the arithmetic unit while also controlling the multiplexer to forward the correct result.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation | Add | Substract | Multiply | Divide |
| op\_code[1:0] | 00 | 01 | 10 | 11 |

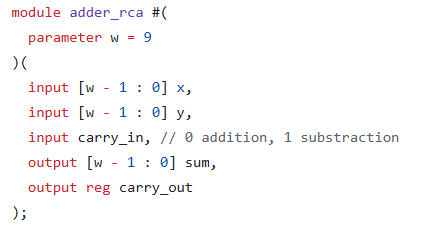
* 1. *Algorithm choices justification:*

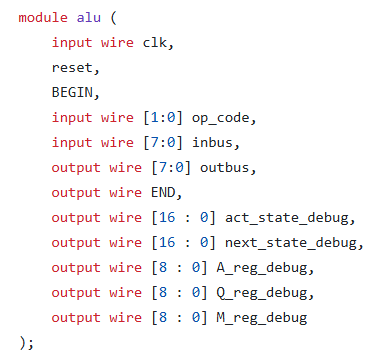
Booth’s Radix-4 multiplication algorithm is selected because it reduces the number of partial products compared to traditional multiplication methods. By grouping the bits in sets of three, the algorithm can handle more significant bit shifts in each cycle, resulting in faster execution. This approach significantly reduces the number of operations, especially for larger bit-width multiplications. For the 8-bit multiplication used in this design, the count is limited to 4 cycles, which helps further reduce the time required for processing the multiplication. The Radix-4 scheme is particularly beneficial as it minimizes the complexity of the multiplier, making it suitable for hardware implementations where both speed and area are important considerations.

The SRT Radix-2 division algorithm is chosen for its ability to strike a balance between computational complexity and speed. It computes the quotient digit in each iteration using simple comparisons and subtractions, which is efficient for small bit-widths like 8 bits. This method provides a straightforward way to implement division, especially in a structural hardware design. By using only basic arithmetic operations and bit shifts, SRT-2 division is well-suited for the target 8-bit design, offering fast division while maintaining manageable hardware complexity.

1. HDL code listings.

This section includes selected HDL code snippets illustrating key architectural components of the 8-bit ALU system. The full Verilog code for all modules is available in the attached .v files.

This module defines a parametrized Ripple Carry Adder (RCA), allowing flexibility for operations over different bit-widths. The design also includes a carry\_in input, used both for propagating carry and as a control flag to indicate whether an addition or a subtraction operation is to be performed.



The ALU module includes standard control signals such as clock, reset, and begin, along with the inbus for operand input and an op\_code signal to specify the desired operation. Output ports consist of the end signal, indicating operation completion, and the outbus, which carries the final result.

Managing the internal sequencing based on the selected operation, this module receives clock, reset, begin, and end signals, and generates various control signals, such as register enables, shift controls, and sign bit management (e.g., for A, Q, and M). It orchestrates the FSM-based control flow of the ALU.

These listings are intended to provide insight into the modular structure and implementation style. For complete code and synthesis-ready files, please refer to the annexed Verilog source files.

1. Testbench description and simulation waveforms.

The testbench is designed to verify the functionality of the ALU by applying various input values and operation codes. Operations tested include addition, subtraction, multiplication, and division, each performed with different operand values. The clock, reset, and begin signals are managed to simulate the real-world behavior of the ALU.

The simulation waveform is shown below, illustrating the ALU's response to these operations.

1. Discussion of results: issues encountered and resolutions.

During the development of the 8-bit ALU, several challenges were encountered, particularly in the early stages of the design process. Initially, we attempted to create separate modules for each operation (addition, subtraction, multiplication, and division). While this approach seemed modular, it quickly became apparent that managing multiple distinct modules created unnecessary complexity. Ensuring that each module interacted correctly and had the proper synchronization was difficult, especially in a way that allowed us to predict the ALU's behavior consistently under different conditions.

After evaluating the difficulties and testing the initial design, we decided to integrate all operations into a single ALU module. This decision simplified the overall structure of the design by reducing the number of interconnections between modules and streamlining the control signals. It made the system more efficient, as operation selection became more straightforward and the need for complex routing between separate modules was eliminated. By consolidating the operations, we were able to reduce potential issues arising from signal synchronization and inter-module communication.

Another key challenge during the design process was ensuring proper synchronization of the control signals, particularly when managing the flip-flops for each operation. This aspect of the design required careful attention to detail to avoid unpredictable behavior, especially under different input conditions. The complexity of handling multiple signals led to difficulties in achieving reliable operation at first.

Ultimately, the decision to integrate the operations into one ALU module resulted in a functional and efficient final design. The ALU was able to perform all required operations, including addition, subtraction, multiplication, and division, with improved efficiency and better control logic. The streamlined design also opened up the possibility for easier future modifications or enhancements if needed.