` Arithmetic and Logic Unit

Name: Omkar Bhat
EMP ID: 6090

Introduction:

The Arithmetic and Logic Unit is a basic combinational circuit that forms the core of any digital system, performing basic arithmetic and logical operations. In this project, a parameterized ALU was designed and implemented in Verilog HDL. This implementation uses a wide variety of operations, including arithmetic operations like addition, subtraction, increment, decrement, and multiplication, together with logical operations such as AND, OR, XOR, NOT, NAND, NOR, and XNOR. The ALU also includes shift, rotate, and comparison operations. Operation selection is controlled with a mode signal mode 0 for logical operations and mode 1 for arithmetic operations and a command input. Except for multiplication, which creates a two-cycle delay, most operations take place in a single clock cycle. The ALU is both modular and scalable with parameterization in support of flexible operand widths. The development phase included RTL design, architectural study, code linting, simulation, and good verification through coverage analysis and functional testing. Having synchronous outputs and good functionality, this ALU design provides a generic building block for System-on-Chip and embedded system applications.

Objectives:

- **Design a synthesizable and Parameterized ALU:** Develop a scalable and synthesizable Arithmetic and Logic Unit (ALU) in Verilog that supports configurable operand widths and a variety of arithmetic and logical operations.
- **Verify Functionality Through Simulation:** Validate the ALU's functionality by writing comprehensive testbenches that include directed tests and corner cases, ensuring the output matches expected results for all supported operations and command inputs.
- **Ensure Code Quality and Test Coverage:** Apply Verilog linting tools to maintain coding standards and detect potential issues and utilize code coverage analysis to confirm that all functional paths and logic conditions are working perfectly during simulation.

Architecture:

Design Architecture:

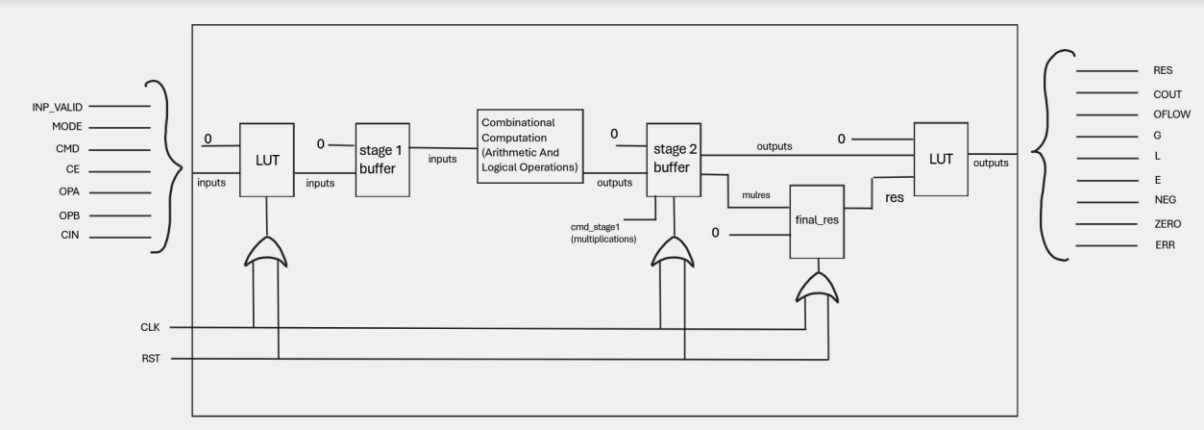


Figure 1: ALU Architecture

The inputs are sampled and stored in a stage_1 input buffer which is sensitive to the positive edge of the clock and reset. Depending on the inputs stored in this buffer the combinational block performs operations and stores the outputs in the stage_2 buffer. For all the operations except for multiplication the outputs are passed to the output ports at the following positive edge of the clock. However, in the case of multiplication operations the outputs are stored in another buffer called final_res and retrieved in the next positive edge to induce an additional delay of one clock cycle.

The inputs to the ALU are described as follows:

- CLK: System clock which is responsible for the synchronous operation of the ALU.
- CE: Clock enable signal. When it is low, the ALU holds its current state.
- RST: Active-high asynchronous reset that clears internal registers and sets all outputs to zero.
- OPA and OPB: Operands for the operation, each parameterized but default to 8 bits.
- INP_VALID: 2-bit input validity signal:
    - MSB corresponds to operand B validity.
    - LSB corresponds to operand A validity.
    - Operations are performed only when respective bits are high, else error is triggered.
- MODE: 1-bit signal to select operation type:
    - 0 - Logical operations
    - 1 - Arithmetic operations

- CMD: A 4-bit command to specify the operation to be performed.

The outputs of the ALU are described as follows:

- RES: Result of the selected operation (operand width + 1 by default, double the width if multiplication).
- OFLOW: Indicates Overflow in the computed result.
- COUT: Indicates carry out or borrow out.
- G, L, E: Indicates the magnitude of operand A against operand B.
- ERR: Indicates if there is any error in the input validity to the command input

Exculsive to the signed operations

- NEG: Indicates if the output result RES is negative.
- ZERO: Indicates if the output result RES is zero.

Testbench Architecture:

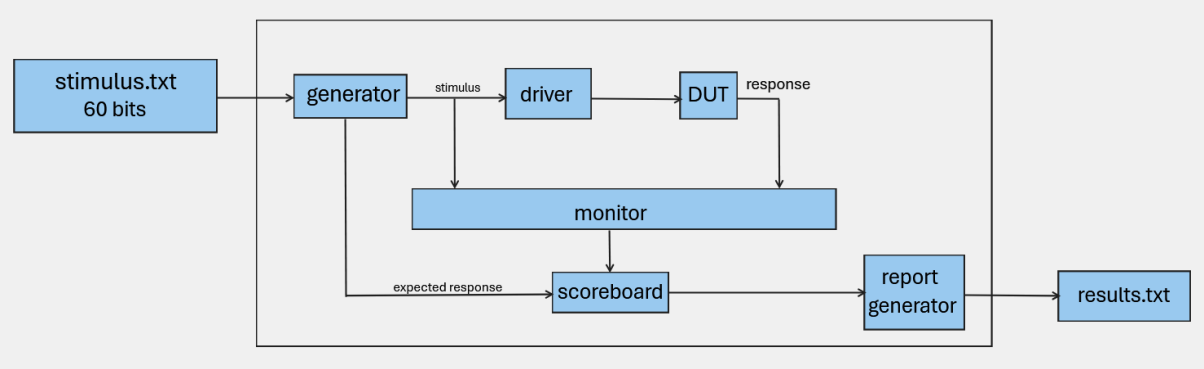


Figure 2: Testbench Architecture

Stimulus Packet Format																
Inputs										Expected Output						
Reserved	Feature ID	RST	CE	MODE	CMD	INP_VALID	OPA	OPB	Cin	RES	COUT	OFLOW	ERR	GLE	NEG	ZERO
3	8	1	1	1	4	2	8	8	1	16	1	1	1	3	1	1
60----58	57----50	49	48	47	46---43	42----41	40----33	32----25	24	23----8	7	6	5	4----2	1	0

Figure 3: Stimulus packet structure

Response Packet																								
Actual Response							Inputs										Expected Output							
RES	COUT	OFLOW	ERR	GLE	NEG	ZERO	Reserved	Feature ID	RST	CE	MODE	CMD	INP_VALID	OPA	OPB	Cin	RES	COUT	OFLOW	ERR	GLE	NEG	ZERO	
16	1	1	1	3	1	1	3	8	1	1	1	4	2	8	8	1	16	1	1	1	3	1	1	
84----69	68	67	66	65----63	62	61	60----58	57----50	49	48	47	46---43	42----41	40----33	32----25	24	23----8	7	6	5	4----2	1	0	

Figure 4: Response Packet structure

The testbench architecture consists of a generator that uses memreadb to read a sixty bits stimulus from a stimulus.txt file the structure of which is shown in Figure 3. The generator passes the stimulus to the driver which drives the DUT. The generator also extracts the expected response from the stimulus packet and sends it to the monitor. The DUT's response shown in Figure 4, as well as the stimulus packet is captured by the monitor and sent to the scoreboard for checking if the expected and obtained response are equal. According to this the report

generator generates the report of whether the testcase passed or failed and writes it to a results.txt file.

Working:

1. Input Buffer stage:

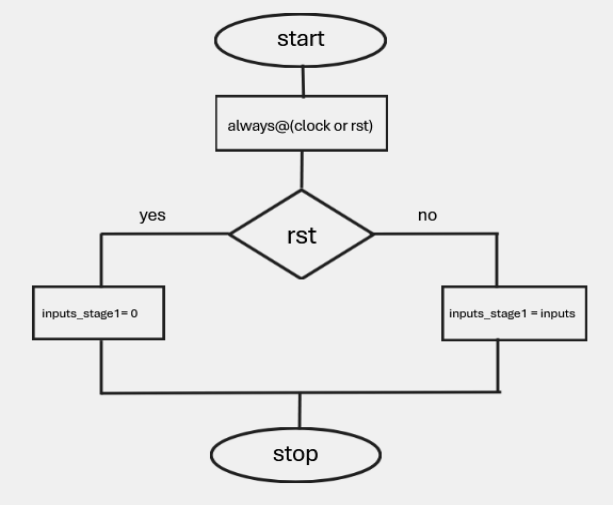
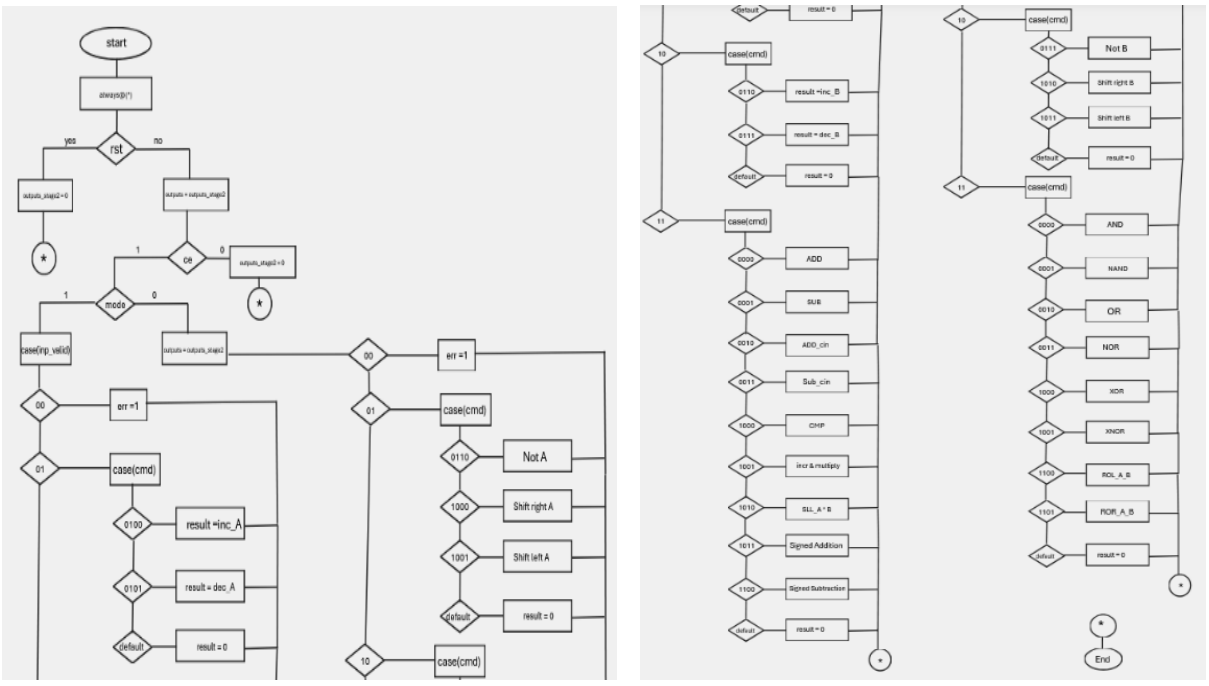


Figure 5: Input buffer stage flowchat

The Input Buffer stage is responsible for capturing input operands and control signals in a synchronized manner before the ALU processes them. This stage operates on the positive edge of the clock and ensures that the inputs are latched when the necessary conditions are met. This ensures that we get one clock cycle of delay between the input sampling and the result of the ALU.

2. Combinational computational block



The logic governing this stage is captured in the flowchart where the combinational block computes the result by accepting inputs from the input buffer. This happens almost instantaneously unlike the updating of buffers that happens for every positive edge of the clock. The ALU samples the input signals under the following conditions:

- The clock enable signal (CE) is asserted high.
- The input valid signal (INP_VALID) indicates that the operands are available, which is a 2-bit vector such that INP_VALID[0] enables operand A, INP_VALID[1] enables operand B and if INP_VALID is 2'b11 in a situation where only one operand is expected, it is considered an invalid condition. In such cases, the error flag (ERR) is asserted, and the output is forced to zero to prevent unintended behaviour.
- The RST is not active.

The MODE signal determines whether the operation to be performed is logical (MODE = 0) or arithmetic (MODE = 1), while the 4-bit CMD signal specifies the exact operation to be executed which are given below:

Arithmetic operations (MODE=1):

- ADD: Unsigned Addition of OPA and OPB. (CMD = 0)
- SUB: Unsigned Subtraction of OPA and OPB. (CMD = 1)
- ADD_CIN: Unsigned Addition of OPA and OPB snd Carry In. (CMD = 2)
- SUB_CIN: Unsigned Subtraction of OPA and OPB snd Borrow In. (CMD = 3)
- INC_A: Increment OPA by 1. (CMD = 4)
- DEC_A: Decrement OPA by 1. (CMD = 5)
- INC_B: Increment OPB by 1. (CMD = 6)
- DEC_B: Decrement OPB by 1. (CMD = 7)
- CMP: Compares OPA and OPB. (CMD = 8)
- INC_MUL: Increments OPA and OPB by 1 and then multiplies them. (CMD = 9)
- SH_MUL: Shifts OPA by 1 and then multiplies OPA and OPB. (CMD = 10)
- SIGN_ADD: Signed addition and comparison of OPA and OPB. (CMD = 11)
- SIGN_SUB: Signed subtraction and comparison of OPA and OPB. (CMD = 12)

Logical Operations (MODE = 0):

- AND: Bitwise AND the two operands. (CMD = 0)
- NAND: Bitwise NAND the two operands. (CMD = 1)
- OR: Bitwise OR the two operands. (CMD = 2)
- NOR: Bitwise NOR the two operands. (CMD = 3)
- XOR: Bitwise XOR the two operands. (CMD = 4)
- XNOR: Bitwise XNOR the two operands. (CMD = 5)
- NOT_A: Inverts OPA. (CMD = 12)
- NOT_B: Inverts OPB. (CMD = 12)
- SHR1_A: Shifts OPA by 1 to the right. (CMD = 12)

- SHL1_A: Shifts OPA by 1 to the left. (CMD = 12)
- SHR1_B: Shifts OPB by 1 to the right. (CMD = 12)
- SHL1_B: Shifts OPB by 1 to the left. (CMD = 12)
- ROL_A_B: Rotates log2(width) bits of OPA by OPB bits to the left. If the OPB bits from the MSB to the log2(width) are active, the error is asserted. (CMD = 12)
- ROR_A_B: Rotates log2(width) bits of OPA by OPB bits to the right. If the OPB bits from the MSB to the log2(width) are active, the error is asserted. (CMD = 12)

3. Output port updation for operations except multiplication.

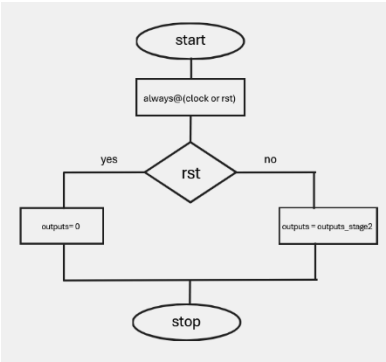


Figure 6: Output port updation

This block updates the output ports for every positive edge of the clock. This enables a two-clock cycle delay for all operations except the ones which perform the multiplication. This stage takes the output stored in the output buffer after the computation by the combinational block and reflects it at the output ports. If a RST signal is asserted, then the outputs and the output buffers are cleared to zero to avoid in discrepancies in computation.

4. Result port updation in the case of multiplication operations

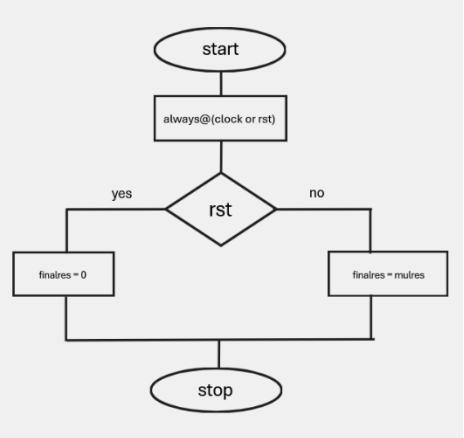


Figure 7: Result updation for multiplication

In the case of CMD 9 and 10 the computed result is supposed to appear in the third clock cycle. For this very reason a separate sequential block is used which reads the output buffer and updates a register called the final_res. This in the following clock cycle reflects the result to the output port thereby adding an external clock cycle delay as required. Even here if the RST signal is asserted then the final_res register is cleared to zero, and the outputs are also cleared.

Result:

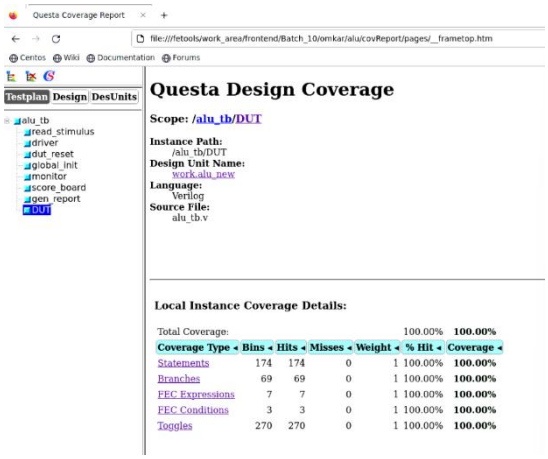


Figure 8: Coverage report from QuestaSim

The ALU was found to perform all the operations correctly and was completely synthesizable. The designed ALU samples the input at the first positive edge of the clock, computes and outputs the result at the second positive edge of the clock in the case operations apart from multiplication. The result of multiplication appears at the positive edge of the third clock cycle. The ALU was tested by passing 55 test cases of which all of them passed. A code coverage of 100 percent was achieved as shown in Figure 8.

Conclusion:

- Successfully designed a parameterized ALU which is completely synthesizable and lint violation free.
- Comprehensively tested and functionally verified using a testbench environment.
- Performed coverage and obtained the reports on the design code.

Future Improvements:

- Make the ALU compatible to work on all input enable signals: If an operation requires both operands to be valid and currently only one operand is valid it waits for the other operand to be valid and then computes the result.
- Make the ALU result width dynamic: Only for the multiplication operations will the result width be twice the width of operands else the result width will be one more than that of the operands.
- Optimizing the code and reducing the complexity: Use more efficient logic and reduce the code lines thereby increasing readability.