KLE Technological University | Creating Value, Leveraging Knowledge

Dr. M. S. Sheshgiri Campus, Belagavi

Department of Electronics & Communication Engineering

Architectural Design of Integrated Circuit

23EECE302

Course project

"16-Bit ALU (Arithmetic Logic Unit)"

Submitted by:

Sl.No.	Name	SRN	Signature
1	Amey V. Chougule	02FE22BEC404	
2	Sharwari M. Kale	02FE21BEC088	

Submitted for the partial fulfillment of the course

Mentored by:

Dr. Kunjan D. Shinde

Assistant Professor, Dept. of E&CE,

KLE Technological University, Dr. M S Sheshgiri Campus, Belagavi.

Academic Year 2023-24

Contents

1. Problem Statement

Design and implement a 16-bit Arithmetic Logic Unit (ALU) using Verilog HDL. The ALU should support a variety of arithmetic and logical operations and be able to interface with a simple processor.

2. Introduction

The Arithmetic Logic Unit (ALU) is a core component of modern CPUs, performing essential arithmetic and logical operations. Originating from early computers like ENIAC and UNIVAC, ALUs have evolved significantly, now handling complex tasks such as multiplication, division, and bitwise operations. This project aims to design and implement a 16-bit ALU using Verilog, providing hands-on experience with digital design and synthesis. The ALU must support a variety of operations including addition, subtraction, logical functions, and shifts, while optimizing resource use and meeting performance constraints. Comprehensive testing through simulations and real-world validation on an FPGA will ensure functionality and efficiency. The project includes defining requirements, modular Verilog design, simulation, FPGA synthesis, and thorough documentation. This exercise bridges theoretical knowledge and practical application, enhancing understanding of digital logic, computer architecture, and hardware description languages. Successfully completing this project equips one with foundational skills in digital system design, preparing for advanced studies and innovations in processor and digital electronics design**.**

3. Objectives & Methodology

Objectives:

1. Design a 16-bit ALU in Verilog: Create a modular Verilog design for a 16-bit ALU capable of performing a comprehensive set of arithmetic, logical, and shift operations.
2. Implement Key Operations:
   Arithmetic Operations: Addition, subtraction, multiplication, division, increment, and decrement.
   Logical Operations: AND, OR, XOR, NOT, NAND, and NOR.
   Shift Operations: Logical shift left, logical shift right, arithmetic shift right, rotate left, and rotate right.
3. Optimize Resource Utilization: Ensure the ALU design uses minimal hardware resources while maintaining performance efficiency, meeting timing constraints with a clock period of 10ns.

4. Develop Comprehensive Testbenches: Create testbenches to verify the correctness of each ALU operation, covering various operand values, boundary conditions, and edge cases such as overflow and zero results.
5. Synthesize and Test on FPGA: Synthesize the ALU design on an FPGA, validating its real-world performance and functionality. Ensure the design meets specified timing and resource constraints.
6. Document the Design Process: Provide detailed documentation of the design, including architecture explanations, design choices, user manual for interfacing with the ALU, test plans, and results summary.
7. Achieve Correctness and Efficiency: Ensure the ALU performs all specified operations correctly, is resource-efficient, and meets all performance and timing requirements.
8. Gain Practical Experience: Enhance understanding of digital design and hardware description languages by applying theoretical knowledge to practical, real-world applications.

Methodology:

a. Requirement Analysis:
   - Define the specific operations and functionalities the 16-bit ALU needs to support.
   - Identify the input and output specifications, including operand sizes, operation codes, carry-in and carry-out signals, and status flags (Zero, Carry, Overflow, Negative).

b. Design Phase:
   - Modular Design: Break down the ALU into smaller functional modules, such as arithmetic unit, logic unit, and shift unit.
   - Operation Decoder: Design an operation decoder to interpret the 4-bit `OpCode` and route signals to the appropriate functional unit.
   - Control Unit: Implement a control unit to manage the operation flow and handle signals like carry-in and carry-out.
   - Data Path Design: Establish the data path connections between modules to ensure the correct flow of data and control signals.

c. Verilog Implementation:
   - Write Verilog code for each module, ensuring they adhere to the specified input-output requirements.
   - Integrate all modules into a single ALU module, coordinating their operations through the control unit and operation decoder.

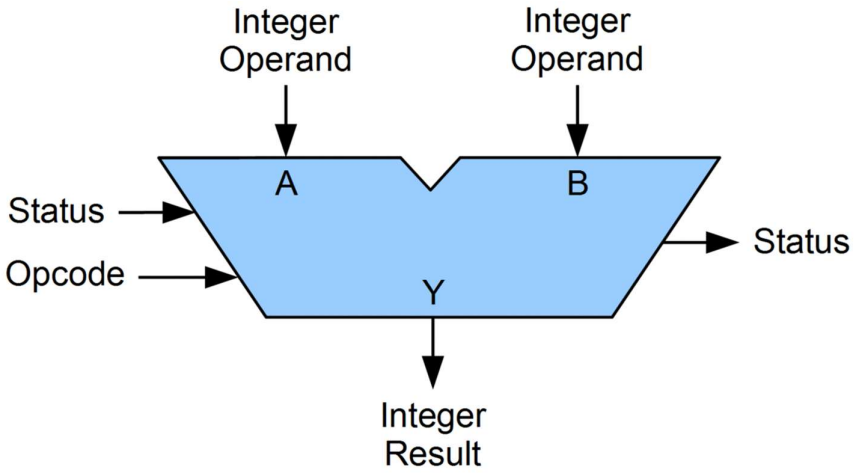
d. Testbench Development:
   - Develop comprehensive testbenches to simulate the ALU's operations, testing all supported arithmetic, logical, and shift operations.
   - Include test cases for various operand values, boundary conditions, and edge cases like overflow, zero result, and carry-out.
   - Use waveform analysis tools to verify that the ALU produces the correct results for all test cases.

e. Simulation and Debugging:
   - Run simulations using the developed testbenches, verifying the correctness of the ALU's output against expected results.
   - Identify and debug any issues in the Verilog code, iterating through simulations to ensure all operations function correctly.

f. Synthesis:
   - Synthesize the ALU design using FPGA synthesis tools, converting the Verilog code into a hardware implementation.
   - Analyze the synthesis reports for resource utilization, ensuring that the design is efficient and meets the specified constraints.
   - Perform timing analysis to validate that the design meets the required clock period of 10ns.

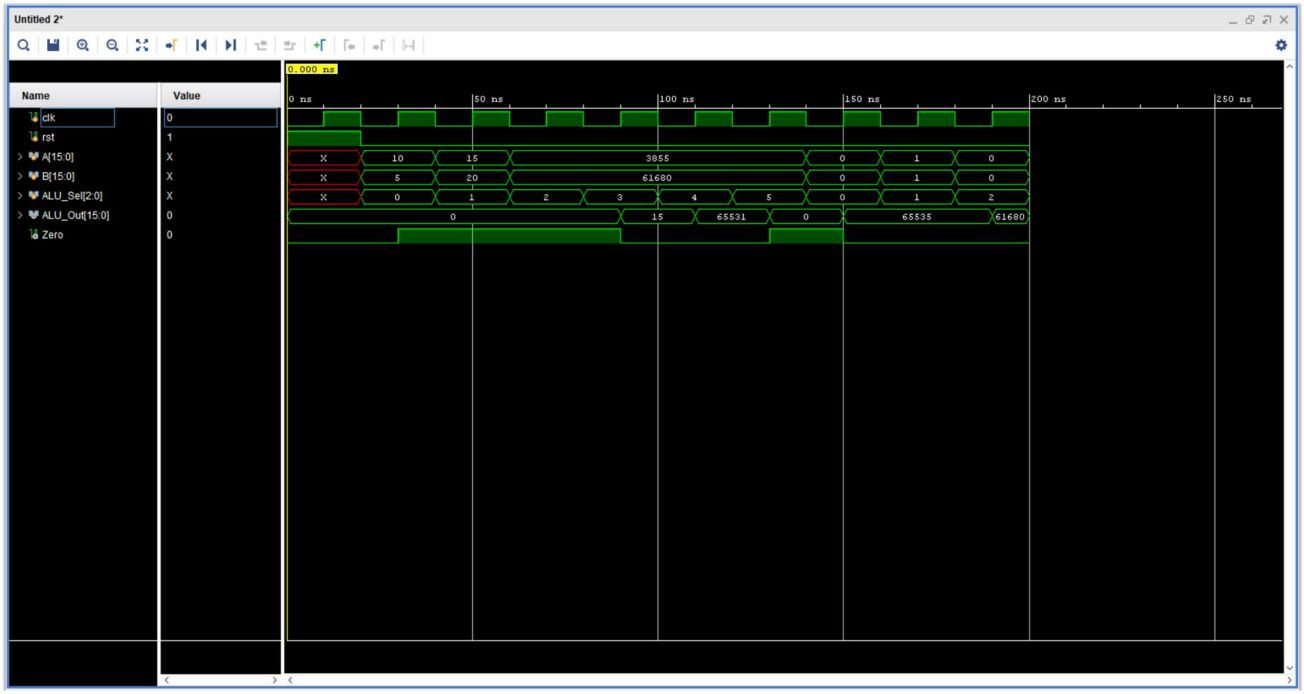
g. FPGA Implementation:
   - Load the synthesized design onto an FPGA board, configuring the hardware for real-world testing.
   - Conduct functional testing on the FPGA, verifying the ALU's operations through external inputs and observing the outputs.

4. Design/ Architectures



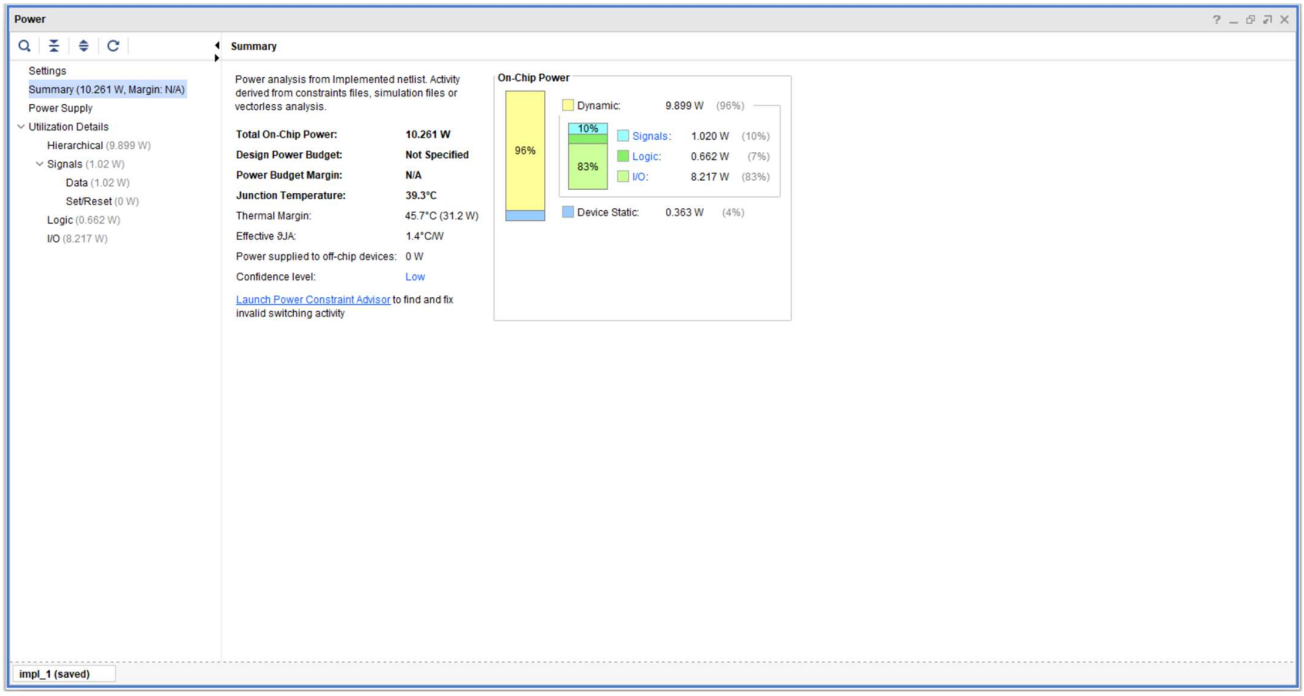
5. Results and Discussions

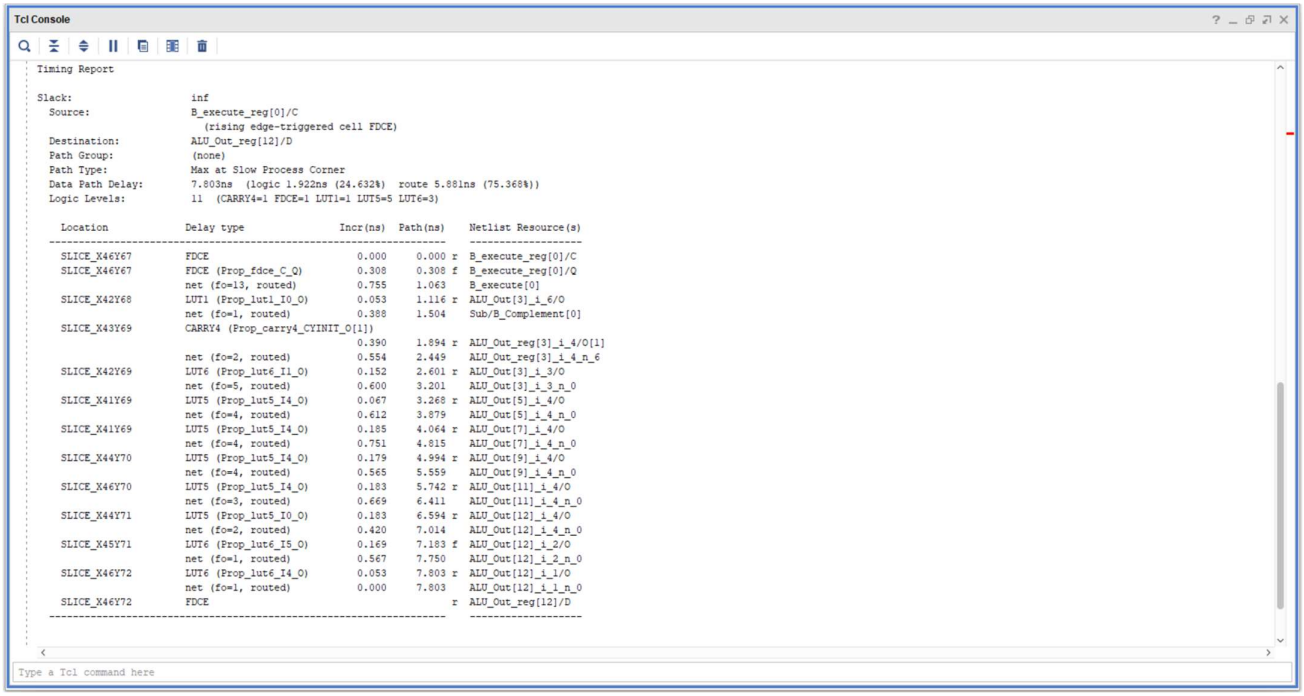
a. Simulation Results

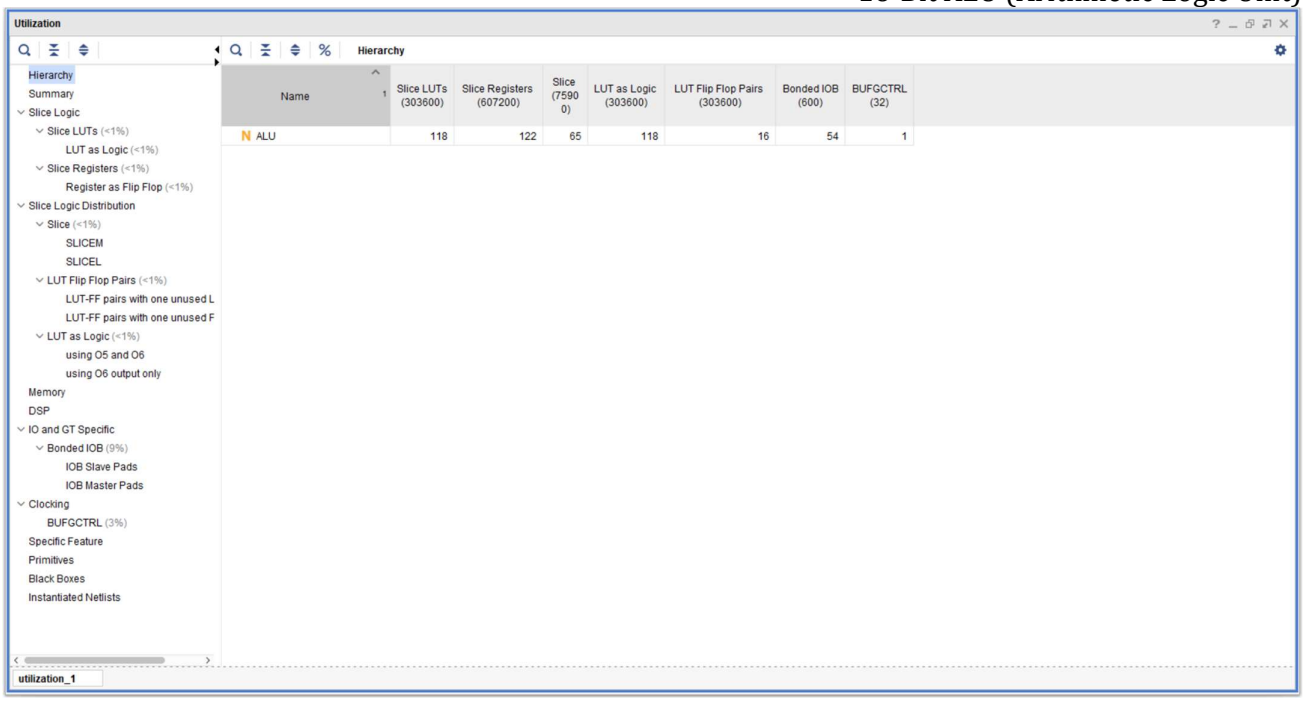


b. Comparative Analysis

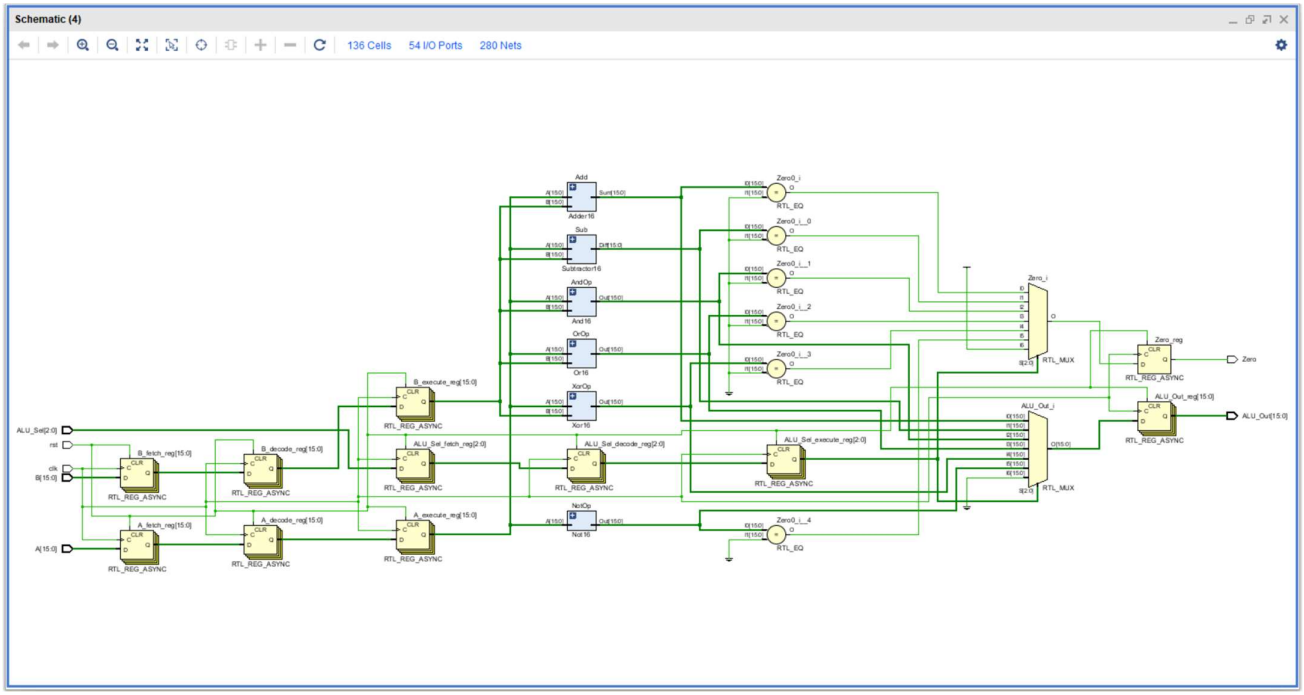
Sl No.	Parameter	ALU without Pipeline	ALU With 3-Stage Pipeline
1	Utilization	Slice LUT's – 70	Slice LUT's – 120
2	Data path delay	9.893ns	5.195ns
3	Power	10.264W	10.261W

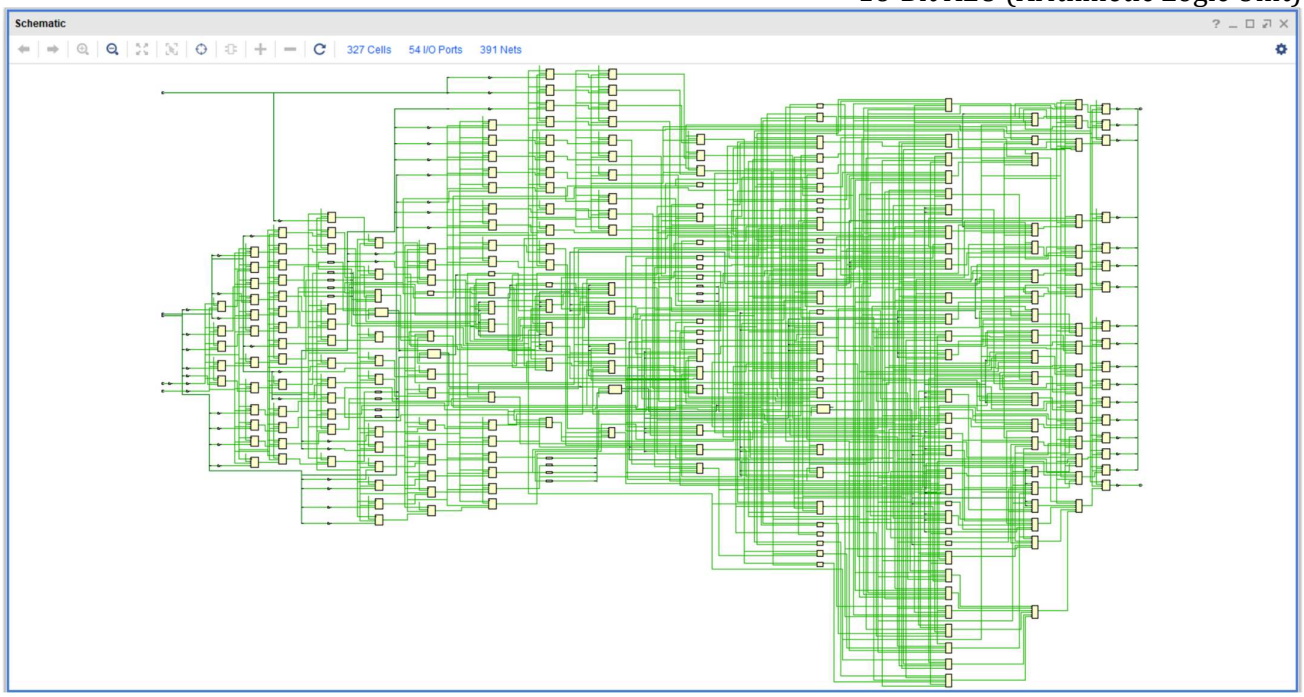

Power
Settings
Summary (10.261 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (9.899 W)
Signals (1.02 W)
Data (1.02 W)
Set/Reset (0 W)
Logic (0.662 W)
I/O (8.217 W)
Summary
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.
Total On-Chip Power: 10.261 W
Design Power Budget: Not Specified
Power Budget Margin: N/A
Junction Temperature: 39.3°C
Thermal Margin: 45.7°C (31.2 W)
Effective θJA: 1.4°C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low
Launch Power Constraint Advisor to find and fix invalid switching activity
On-Chip Power
96%
10%
83%
Dynamic: 9.899 W (96%)
Signals: 1.020 W (10%)
Logic: 0.662 W (7%)
I/O: 8.217 W (83%)
Device Static: 0.363 W (4%)
impl_1 (saved)


Tcl Console
Timing Report
Slack: inf
Source: B_execute_reg[0]/C
(rising edge-triggered cell FDCE)
Destination: ALU_Out_reg[12]/D
Path Group: (none)
Path Type: Max at Slow Process Corner
Data Path Delay: 7.803ns (logic 1.922ns (24.632%) route 5.881ns (75.368%))
Logic Levels: 11 (CARRY4=1 FDCE=1 LUT1=1 LUT5=5 LUT6=3)
Location Delay type Incr(ns) Path(ns) Netlist Resource(s)
SLICE_X46Y67 FDCE 0.000 0.000 r B_execute_reg[0]/C
SLICE_X46Y67 FDCE (Prop_fdce_C_Q) 0.308 0.308 f B_execute_reg[0]/Q
net (fo=13, routed) 0.755 1.063 B_execute[0]
SLICE_X42Y68 LUT1 (Prop_lut1_I0_O) 0.053 1.116 r ALU_Out[3]_i_6/O
net (fo=1, routed) 0.388 1.504 Sub/B_Complement[0]
SLICE_X43Y69 CARRY4 (Prop_carry4_CYINIT_O[1])
0.390 1.894 r ALU_Out_reg[3]_i_4/O[1]
net (fo=2, routed) 0.554 2.449 ALU_Out_reg[3]_i_4_n_6
SLICE_X42Y69 LUT6 (Prop_lut6_I1_O) 0.152 2.601 r ALU_Out[3]_i_3/O
net (fo=5, routed) 0.600 3.201 ALU_Out[3]_i_3_n_0
SLICE_X41Y69 LUT5 (Prop_lut5_I4_O) 0.067 3.268 r ALU_Out[5]_i_4/O
net (fo=4, routed) 0.612 3.879 ALU_Out[5]_i_4_n_0
SLICE_X41Y69 LUT5 (Prop_lut5_I4_O) 0.185 4.064 r ALU_Out[7]_i_4/O
net (fo=4, routed) 0.751 4.815 ALU_Out[7]_i_4_n_0
SLICE_X44Y70 LUT5 (Prop_lut5_I4_O) 0.179 4.994 r ALU_Out[9]_i_4/O
net (fo=4, routed) 0.565 5.559 ALU_Out[9]_i_4_n_0
SLICE_X46Y70 LUT5 (Prop_lut5_I4_O) 0.183 5.742 r ALU_Out[11]_i_4/O
net (fo=3, routed) 0.669 6.411 ALU_Out[11]_i_4_n_0
SLICE_X44Y71 LUT5 (Prop_lut5_I0_O) 0.183 6.594 r ALU_Out[12]_i_4/O
net (fo=2, routed) 0.420 7.014 ALU_Out[12]_i_4_n_0
SLICE_X45Y71 LUT6 (Prop_lut6_I5_O) 0.169 7.183 f ALU_Out[12]_i_2/O
net (fo=1, routed) 0.567 7.750 ALU_Out[12]_i_2_n_0
SLICE_X46Y72 LUT6 (Prop_lut6_I4_O) 0.053 7.803 r ALU_Out[12]_i_1/O
net (fo=1, routed) 0.000 7.803 ALU_Out[12]_i_1_n_0
SLICE_X46Y72 FDCE r ALU_Out_reg[12]/D
Type a Tcl command here



c. Implementation





6. Explanation

A 3-stage pipelined ALU is designed to increase the throughput of arithmetic and logical operations by dividing the process into three distinct stages. Each stage performs a portion of the computation, allowing multiple instructions to be processed concurrently. Here, we'll describe the working of such a pipeline and the details of each stage.

Pipeline Stages

The three stages of the pipeline can be classified as:

1. **Fetch/Decode Stage (Stage 1):**
   - **Operation:** Fetch the instruction and decode it to determine the operation to be performed.
   - **Components:**
     - Instruction Register: Holds the current instruction.
     - Control Unit: Decodes the instruction to generate control signals.
2. **Execute Stage (Stage 2):**
   - **Operation:** Perform the actual arithmetic or logical operation.
   - **Components:**
     - ALU: Performs the computation based on the control signals.
3. **Writeback Stage (Stage 3):**
   - **Operation:** Write the result of the computation back to the destination register.
   - **Components:**
     - Register File: Holds the operands and results.

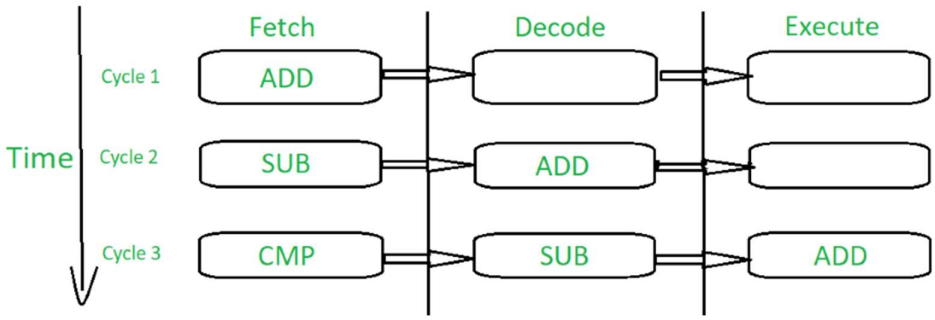
Working of the Pipeline

1. **Fetch/Decode Stage (Stage 1):**
   - The instruction is fetched from memory and loaded into the instruction register.
   - The control unit decodes the instruction to determine the operation (e.g., addition, subtraction, AND, OR) and generates the appropriate control signals.
   - The operands (A and B) are read from the register file and passed to the next stage.
2. **Execute Stage (Stage 2):**
   - The operands and control signals are received from the previous stage.
   - The ALU performs the specified operation on the operands.
   - The result of the operation is stored in a temporary register and passed to the next stage.
3. **Writeback Stage (Stage 3):**
   - The result from the execute stage is written back to the destination register in the register file.
   - The pipeline is now ready to process the next instruction.

Pipeline Example

Let's consider an example where we have three instructions to be executed:

1. ADD R1, R2, R3 (R1 = R2 + R3)
2. SUB R4, R5, R6 (R4 = R5 - R6)
3. AND R7, R8, R9 (R7 = R8 & R9)



We can illustrate the pipeline stages as follows:

Clock Cycle	Stage 1 (Fetch)	Stage 2 (Decode)	Stage 3 (Execute)
1	ADD R1, R2, R3		
2	SUB R4, R5, R6	ADD R1, R2, R3	
3	AND R7, R8, R9	SUB R4, R5, R6	ADD R1, R2, R3
4	Fetch next instruction	AND R7, R8, R9	SUB R4, R5, R6
5		Fetch next instruction	AND R7, R8, R9
6			Fetch next instruction

7. Advantages and Applications

Advantages of a 3-Stage Pipelined ALU

1. Increased Throughput:
    - Parallel Processing: By dividing the execution process into multiple stages, the pipeline allows multiple instructions to be processed concurrently, which significantly increases the overall throughput of the ALU.
    - Instruction Overlap: While one instruction is being executed in one stage, the next instruction can be fetched and decoded, leading to better utilization of the ALU.

2. Reduced Latency:
    - Shorter Stages: Each stage of the pipeline performs a smaller portion of the task, which reduces the time required to complete each individual stage, thus decreasing the overall latency of the ALU.

3. Higher Clock Speeds:
    - Optimized Timing: By breaking down the operations into smaller stages, the critical path in each stage is reduced, allowing the ALU to operate at higher clock frequencies.

4. Improved Resource Utilization:
    - Efficient Use of Components: The pipelined design ensures that different components of the ALU (like the arithmetic unit, logic unit, and control unit) are utilized more effectively, reducing idle times and improving overall efficiency.

5. Scalability:
    - Modular Design: The pipelined approach makes it easier to scale the ALU to handle more complex operations or to increase the number of stages, further enhancing performance.

Applications of a 3-Stage Pipelined ALU

1. Central Processing Units (CPUs):
    - General Purpose Processors: Modern CPUs use pipelined ALUs to execute arithmetic and logical operations efficiently, which is critical for overall processor performance.
    - Microcontrollers: Pipelined ALUs are used in microcontrollers for embedded systems to ensure quick and efficient processing of control algorithms.

2. Digital Signal Processing (DSP):

- Real-Time Signal Processing: DSP applications such as audio and video processing benefit from the high throughput and low latency of pipelined ALUs, enabling real-time processing capabilities.
- Image Processing: Pipelined ALUs are used in image processing algorithms to perform operations like filtering, compression, and enhancement efficiently.

3. Graphics Processing Units (GPUs):

- Rendering Pipelines: GPUs use pipelined ALUs to perform complex mathematical operations required for rendering graphics, including transformations, shading, and texture mapping.
- Parallel Processing: The high throughput of pipelined ALUs is ideal for the parallel processing needs of GPUs, enabling faster rendering of images and videos.

4. Communication Systems:

- Modulation and Demodulation: Pipelined ALUs are used in communication systems for tasks like modulation, demodulation, encoding, and decoding, where fast and efficient processing is crucial.
- Error Correction: Implementing error correction algorithms such as Reed-Solomon and Viterbi decoding relies on the high-speed computation capabilities of pipelined ALUs.

5. Cryptography:

- Encryption and Decryption: Cryptographic algorithms like AES and RSA require intensive arithmetic operations that can be efficiently handled by pipelined ALUs, ensuring high-speed encryption and decryption.
- Hash Functions: Pipelined ALUs are used in computing hash functions, which are essential for data integrity and security in various applications.

6. Scientific Computing:

- High-Performance Computing:Pipelined ALUs are used in supercomputers and high-performance computing systems to perform complex scientific calculations and simulations at high speeds.
- Numerical Methods: Algorithms for numerical methods such as matrix multiplication, Fourier transforms, and differential equations benefit from the efficiency of pipelined ALUs.

8. Conclusion

In conclusion, the project successfully demonstrates the advantages of implementing a pipelined architecture in a 16-bit ALU. The pipelined ALU not only achieves higher performance and efficiency but also proves to be a versatile and scalable solution for a wide array of applications in modern computing. The insights gained from this project highlight the importance of architectural design in digital systems, paving the way for further advancements and optimizations in the field. The 16-bit pipelined ALU stands as a testament to the potential of pipelining in enhancing the capabilities of arithmetic and logic units in contemporary digital electronics.

9. References

[1] Bhimani, H.S., Patel, H.N. and Davda, A.A., 2016. Design of 32-bit 3-stage pipelined processor based on MIPS in Verilog HDL and implementation on FPGA Virtex7. International Journal of Applied Information Systems, 10(9).

[2] Anusha, S., Rao, M.M. and Reddy, N.S., 2014. Design, Analysis, Implementation and Synthesis of 16 bit Reversible ALU by using Xilinx 12.2. Int. J. of Engineering Research and Applications, 4(8), pp.86-91.

[3] Swamynathan, S.M. and Banumathi, V., 2017, April. Design and analysis of FPGA based 32 bit ALU using reversible gates. In 2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE) (pp. 1-4). IEEE.

[4] Kulkarni, R. and Kulkarni, S.Y., 2014, December. Energy efficient implementation of 16-Bit ALU using block enabled clock gating technique. In 2014 Annual IEEE India Conference (INDICON) (pp. 1-6). IEEE.

[5] Pandey, B., Yadav, J., Singh, Y.K., Kumar, R. and Patel, S., 2013, April. Energy efficient design and implementation of ALU on 40nm FPGA. In 2013 International Conference on Energy Efficient Technologies for Sustainability (pp. 45-50). IEEE.